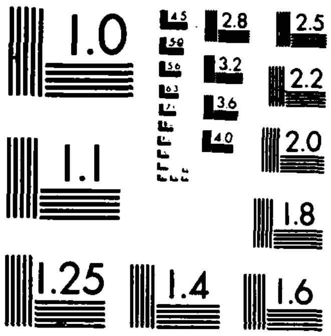
AD-A174 268 ADDRESSING FOR MOBILE SUBSCRIBERS IN A PACKET-SWITCHING 1/2
MULTI-NETWORK ENVIRONMENT(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI G M NUSZ
UNCLASSIFIED DEC 84 AFIT/GCS/ENG/84D-19 F/G 17/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

AFIT/GCS/ENG/84

ADDRESSING FOR MOBILE SUBSCRIBERS
IN A PACKET-SWITCHING
MULTI-NETWORK ENVIRONMENT

THESIS

Gail M. Nusz
Captain, USAF

AFIT/GCS/ENG/84D-19

Approved for public release; distribution unlimited

AFIT/GCS/ENG/84D-19 ✓

ADDRESSING FOR MOBILE SUBSCRIBERS IN A PACKET-SWITCHING MULTI-NETWORK ENVIRONMENT

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

Gail M. Nusz, B.S.
Captain, USA.

December 1984

Approved for public release; distribution unlimited

Acknowledgments

In performing the research, coding the simulation, and writing this thesis I have had a great deal of help from others. I am deeply indebted to my faculty advisor, Major Walter D. Seward, for his encouragement and assistance in times of need. I also wish to thank my reader, Captain Gary F. Thomas (US Army), for his assistance in the final editing process. Finally, I wish to thank Captain Glen B. Ledeboer (US Army) for his assistance in debugging my computer code and his help in the editing and reediting of the writing.

Gail M. Nusz

Table of Contents

List of Figures

List of Tables

Abstract

This thesis addresses the problem of maintaining the TCP-based applications of a subscriber in a mobile environment. Two solutions to the problem of addressing these mobile subscribers has been suggested. The first scheme of addressing is to use logical addresses at the internet level and the second is to use separate addresses at the transport level of the protocol hierarchy. The performance of the two proposed solutions are analyzed through the use of a simulation program based on queueing models.

Addressing for Mobile Subscribers in a Packet-Switching Multi-Network Environment

I. Introduction

Background

A mobile subscriber is actually a host computer that travels around in a network or from one network to another. The development of packet radio technology has been stimulated by the need to provide computer network access to mobile subscribers. Packet radio offers a highly efficient way of using a multiple access channel, particularly with mobile subscribers. Good connectivity can be maintained in a packet radio network with mobile subscribers as long as a line of sight path exists, or a system of relays is used to complete the transmission of data. [1]

Packet switching networks use a hierarchy of protocols for the transfer of packets of information among network subscribers. The transmission control protocol (TCP) and the internet protocol (IP) used in DoD data networks at the transport level are a suitable choice to be used in the environment of this study. TCP/IP interfaces on the lower level with the communications network and on the upper level to the user or application process. This relationship can be seen in figure 1. TCP/IP are expected to operate within military environments where deliberate physical destruction and electronic interference aimed at the communication systems simply must not degrade essential network services. Connections are opened from one subscriber to another by TCP using an internetwork address to partially define the connection. This address is shared and also used by IP for routing packets. [2]

USER
TCP/IP
NETWORK
LINK
PHYSICAL

Figure 1. Protocol Hierarchy

Problem

This thesis addresses the problem of maintaining the TCP-based applications of a subscriber in a mobile environment. That is, can a subscriber keep TCP (virtual) connections open while moving around an internet system? The specific question that needs to be answered on a general level is: how would a host carried on a helicopter manage to preserve its TCP-based applications while it flies out of radio contact with one packet radio and into contact with another packet radio net?

Scope

Possible solutions to be explored in the problem of addressing mobile subscribers will be limited to those using logical addresses at the internet level and those using separate addresses at the transport level of the protocol hierarchy.

Assumptions

The problem will be generalized by considering the host computer being carried not only on a helicopter and the contraints it imposes, but that the host may be carried in a fixed-wing aircraft or on-board a ship.

Summary of Current Knowledge

In this section, a review of the current literature will include the areas of packet radio technology, some of the problems encountered in having mobile subscribers in multi-network systems, and how Transmission Control Protocol (TCP) / Internet Protocol (IP) function.

"Packet radio is a technology that extends the original packet switching concepts which evolved for networks of point-to-point communication lines to the domain of broadcast radio networks." [1:1468] A packet radio network is appropriate for use in a hostile environment with mobile subscribers because of its rapid and convenient deployment capability, survivability, and ease of reconfiguration. In this application, packet radio is also appropriate because of its ability to support mobile users over a large geographical area. The area coverage is easily expandable and the number of users connected may also change. Packet radio is also capable of existing with other packet radio networks, and is therefore appropriate for use in a multi-network environment. [3:99]

Supporting real-time interactions between subscribers is the primary objective of a packet radio network. In order to satisfy this requirement, there are several basic functions a packet radio network must provide. These basic functions can be divided into two subgroups: those which are automatically provided and those which a subscriber may ask for specifically. The first group includes functions such as the following:

1. Network transparency - the subscriber should only have to give a destination address and not have to be concerned with routing, reliability, and other networking concerns.

2. Area coverage and connectivity - the network should allow area coverage to be expanded at the expense of end-to-end delay. The network should allow any subscriber to connect with any other subscriber.

3. Mobility - the system should support mobile subscribers.

4. Internetting - the network should be capable of routing packets for another network to the point of connection with that other network.

5. Throughput and low delay - the network should allow for variable length packets which should be delivered with very short delays to maintain the real-time interactive ability.

6. Rapid and convenient deployment - deployment should require no more than mounting the equipment. The network should discover the connectivity between subscribers. In other words, the system should be self-initializing.

The second group, those which the subscriber must ask for specifically, includes functions such as:

1. Error control - error control should be provided by the network, although the subscriber should have the option of what method should be used for correction.

2. Routing options - the subscriber should be able to specify the type of routing used through its parameters. For example, the subscriber may choose either point-to-point or broadcast as the type of routing to be used.

3. Addressing Options - the network should provide a capability for addressing a subset of subscribers.

4. Tactical Applications - in tactical military operations the packet radio should provide resistance to jamming, detection, and direction finding. [1]

The network topology is a dynamic topology in an environment where the subscribers are frequently or constantly changing their locations. Performance limits are imposed on the system by the presence of mobile subscribers. Where point-to-point routing is used, the maximum allowable speed for mobile subscribers is limited by the time it takes the system to adapt to changes along a subscribers path.There are several methods for routing packets of data in such a network topology. Two such methods which will be discussed here are a broadcast routing algorithm and an algorithm which utilizes a base station. [4,5]

In a broadcast network the major decision is whether to accept a packet or to reject it, not to determine which outgoing line to use as is done in a point-to-point network (such as the ARPAnet).

A broadcast routing algorithm transmits the packet to all subscribers in the network. Each subscriber determines whether that packet was intended for it, whether to forward the packet, or whether to discard the packet. Broadcast routing is very practical in networks where subscribers are mobile and locations are not always known. No addressing is used in this scheme, a subscriber recognizes packets sent to it by comparing the destination ID against its own ID. [6:926-927]

The basic network model where subscribers cannot rely on line-of-sight transmission from sender to receiver utilizes a base station which relays the packets from sender to receiver. The sender transmits a packet to the base station. The base station then signals to all subscribers that a packet is available to be transmitted. The destination subscriber recognizes that the packet is addressed to it and signals to the base stations that it is ready to receive the packet. The base station then relays the packet to its destination subscriber, thus completing the transmission. [7]

The next topic of this section deals with the interconnecting of networks. To allow subscribers in different networks to communicate with each other, a basic set of problems such as addressing and routing procedures, must be solved. The problem of addressing in a mobile environment are discussed in Chapter 2.

One approach to addressing and routing in large systems is to use hierarchical methods. A switch serves a subset of the subscribers in transmitting packets. If every switch maintained routing information for every destination subscriber, the routing tables would become very large. To reduce routing table size, the method of hierarchical addressing is suggested. Each network has a number of switches. Each switch has a number of ports, one for each subscriber serviced by that switch. Each address would then contain a network address, a switch address, and a port address <net,switch,port>. Routing may be done by sending packets for any subscriber in a particular net in the switch over the same route, thus reducing the number of entries in the routing table. The reduction of routing tables has its price; the resulting routes may not always be optimal [5]. Thus it can be seen that some solutions to some problems may result in other problems.

The ARPA sponsored work on networks has resulted in the development of an internet protocol (IP) and a transmission control protocol (TCP) to handle the problems encountered in the interconnection of networks. Networks are interconnected by gateway computers that appear to be local subscribers on two or more nets. The gateways are responsible for routing traffic across multiple networks, and for forwarding messages across each net using the packet transmission protocol in each network. This approach allows the interconnection of networks that have significantly different internal protocols and performance. Gateways provide an internet service by means of IP. The format of the internet packets and the rules for performing internet protocol functions based on the control information in the packets is provided by IP. IP must be implemented in subscribers engaged in internet communication as well as in the gateways. [8] A more detailed look at IP and TCP follows.

TCP/IP developed as the direct result of the evolution of the DoD Internet Architecture Model as described by Cerf and Cain. [9] TCP developed as a highly reliable, logical connection, end-to-end transport protocol which would reside within each host. IP developed as a very simple datagram protocol which would reside in the hosts as well but would also provide the internetwork protocol within the more vulnerable gateways and IMP's. [10:607]

The TCP is a highly reliable, end-to-end, logical connection transport level protocol which logically functions within a packet switched environment between hosts on a single network or between hosts in an internetwork system. TCP makes few assumptions concerning any of the layers below itself other than that a simple datagram service exists which is probably unreliable. [2:1] In this case the simple datagram service is IP. Consequently, TCP must perform in the following areas [2:3]:

Basic Data Transfer
Reliability
Flow Control
Multiplexing
Connections
Precedence

TCP's high reliability is due to the assignment of a unique sequence number to each octet (i.e., 8 bits) transmitted and to the use of a positive acknowledgement scheme between hosts [2:4]. TCP controls the flow of segments via a window technique much as that described by Tannenbaum [11:148-164] at the data link layer. A window is sent back with each ACK containing a maximum sequence number that a host is willing to accept. By subtracting the last sequence number sent from the maximum sequence number received in an ACK segment, a host can determine the number of octets he is subsequently allowed to transmit. TCP allows processes within a host to multiplex on the communications lines attached to the hosts by providing sockets which any process may use. A socket is composed of the concatenation of a host communications port address and the internetwork and host address. A pair of sockets would then uniquely identify a connection between two processes. Finally, the TCP security and Precedence function is essentially enacted through the internet layer below it. A more detailed description of each of these functions can be found in [2].

The Connections function of the TCP is the management of establishing, maintaining and closing communications between two processes residing on two different hosts. A connection is established between two processes when a pair of sockets has been identified, initial sequence numbers have been exchanged and window sizes have been established. TCP uses a three way handshake to perform these tasks. [2]

Once a connection is established, the users may exchange data. Since TCP uses a positive acknowledgement scheme each data segment must carry an ACK. Also, because TCP is designed to be highly reliable, a checksum is calculated for all octets within the data. Retransmissions, based on a timeout, are sent until ACK is received guaranteeing deliver of every segment. [2:40] This data exchange cycle continues until a close request is initiated.

TCP, in order to implement these functions, prefixes the data transmitted with a TCP header as shown in Fig. 3. The header is a sequence of 32-bit words with up to a maximum of 6 words (variable due to the options) per segment. A complete definition of each of the fields shown in Fig. 2 is given in [2].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2. TCP Header Format

The IP is a simple internetwork, datagram protocol which provides no end-to-end services but merely offers a datagram to the communications network and forgets it. The IP Standard succintly describes what the protocol is and is not: "The internet protocol implements two basic functions: addressing and fragmentation. ...The internet protocol does not provide a reliable communiction facitlity. There are no acknowledgments wither end-to-end or hop-by-hop. There is no error control for data only a header checksum. There are no retransmissions. There is no flow control." [12:2-3]

The IP Standard makes a distinction between names, addresses and routes. Basically, the IP uses addresses to define where a named host resides in the network. IP

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3. IP Header Format

simply translates between names and addresses to allow the lower levels to correctly route the datagrams [12:7]. The IP's fragmentation function allows arbitrarily long (max 64K octets) segments to traverse a number of networks whose segment lengths may vary. Like TCP these functions depend on a header prefixed to transmitted data. Fig. 3 depicts the IP header. The IP Standard contains full descriptions of all these fields [12].

As we have seen in this brief survey of a packet radio multi-network environment with mobile subscribers that packet radio is an appropriate choice in military applications. We have also seen that there are many problems involved in allowing subscribers to be mobile. Problems exist in addressing and routing within a communication network with or without mobile subscribers. From this brief survey one can see that much research remains to be done in the areas of routing and addressing to improve efficiency and to reduce costs in a packet radio internetwork with mobile subscribers.

Approach

In this thesis, a summary of current literature is presented of specific definitions and proposed solutions to the model radio networking problem. Next, the possible solutions of using logical addresses at the IP level or using separate addresses at the TCP level will be explored. Lastly, a cost/benefit analysis of the above solutions will be done. The method used to explore these solutions and to conduct a cost/benefit analysis is that of a computer simulation of the mobile subscriber environment described in the background section. The results of this computer simulation is used to compare the two proposed solutions.

Thesis Overview

Chapter 2 contains the results of the literature search performed to identify previous problem definition and the proposed solutions. Chapter 3 describes the two techniques of addressing which are evaluated as solutions to the problem as stated above in the problem section. Chapter 4 contains a discussion of the computer simulation. Included in this chapter is the development of the models used to represent each proposed solution. Chapter 5 contains the results of the computer simulation including the cost/benefit analysis of each of the proposed solutions. Chapter 6 describes the results and conclusions of this research and possible extensions of this effort.

II. Addressing Mobile Subscribers

The results of the literature search performed to identify previous definitions of the problem of addressing mobile subscribers and the proposed solutions to the problem are presented in this chapter. Addressing a subscriber in a network with only nonmobile subscribers does not include the problem of maintaining a connection as the physical address, or location, of the subscribers will not change. The problem of addressing mobile subscribers occurs when the subscribers change their physical address. In this discussion the term "node" is used to refer to the switching computer in the network, and "subscriber" is used to refer to the host computer or terminal equipment connected to the network. Connection of the subscriber to the node is over a communications circuit terminating at a port on the node. A subscriber can be addressed by the node number and port number to which it is connected within a given network; this is referred to as "physical addressing". [13] A brief discussion of what an address is and where it fits into the scheme of things is presented.

Names, Addresses, Routes

Names, addresses, and routes are three distinct entities in an internet. The name of a resource indicates what we seek, an address indicates where it is, and a route tells us how to get there. [14]

A name is a symbol identifying a subscriber. A mechanism exists to map names into addresses. The way in which this mapping mechanism functions depends on the choice of implementation. The mapping process may be handled by a central controller or the mapping process may be handled through a distributed system of controllers. The advantage of a central controller is that changes in the addresses of subscribers only have to be made at the central controller. A disadvantage to a central controller is a delay in the mapping process due to contention at the central controller. An advantage to the distributed system besides faster response, is that of survivability. If the central controller is inoperable for any reason, no mapping can take place in a system with only one controller. A disadvantage to a system of distributed controllers is that of updating changes in a timely manner. The name is not bound to the address until the mapping takes place; thus allowing the address associated with the name of a particular subscriber to change over time.

The address is the data structure which defines the subscriber and can be recognized by all elements. Addresses must be meaningful and must be drawn from some uniform address space. The address space may be either a "flat" one which spans the entire domain or it may be a heirarchical address space such as that used in the Pup internet. In a flat address space the address is not divided into any subfields such as the social security number which is a consecutive nine digit field. In the Pup internet, the address consists of three parts: a network number, a host number, and a socket number [15:614].

When one subscriber wishes to communicate with another, the address of the destination subscriber is mapped into an appropriate route. The address need not be bound to a route until the mapping process takes place allowing the route to change over time. A route is the specific information needed to forward a packet to its destination subscriber.

The route mapping may be done at the source or at intermediate points along the route. In source routing, decisions for routing from source to destination are made before the data leaves the source. In intermediate routing, a decision is made at every switching point, only the destination address is carried along with the data.

In this thesis, the process of mapping a name into an address is studied. The rest of this chapter covers suggested solutions to the problem of addressing mobile subscribers.

Specialized Database

In the method suggested by Schoch, 1978, mobile subscribers are considered to represent a special case of the multiple address problem. A multiple address occurs when a subscriber may have two different addresses within a network or be connected to two different networks. The suggested solution is to maintain a specialized database of the mobile subscribers current locations within individual packet radio nets. The mobile subscribers send updates to the database when changes occur. Subscribers wishing to communicate with the mobile subscribers would query the database for the current address. [4:14] One disadvantage to this solution, in adddition to its vulnerability, is the need for a separate communication with the database which must be successful before a connection with a mobile subscriber can be made. Another complication to this solution is that as a mobile subscriber changes networks, updates to the database must be made in a timely manner. [8]

Virtual Nets

The problem of allowing subscribers to move between nets is that the network portion of the internet address is interpreted as specifying the "real" network in which the subscriber must be found. A subscriber moving to a new network would have to take on a new internet address, causing problems for higher level protocols, such as TCP, that use the internet address for identifying what connection a packet belongs to. The suggested solution is one in which a "virtual" net exists and source routing is used. A "virtual" net does not correspond to a physical net, but rather to a set of subscribers in which names can be uniquely assigned. There would be one network identifier for mobile subscribers where the local portion of the internet address for each packet radio would be a unique 24-bit number. The network address must then be interpreted in a different manner in internet routing. A two-part source route must be used to deliver packets to airborne subscribers. The first step would be a normal internet address of a forwarder in the net the source subscriber is in. The mobile virtual net address in the final part of the source route would cause the forwarder to look up in a dynamically maintained table of attached mobile subscribers what the proper local address was to the specified mobile subscriber, and forward the packet accordingly. This solution allows higher level protocols to remain unchanged in their mapping of addresses to connections. This freedom for higher level protocols has its price at the internet level. Source routing is required, forwarders must maintain local address mappings, and an appropriate forwarder for each mobile subscriber must be determined. [5]

Logical Addressing

In many computer networks, the subscriber presents a message to the network with an address corresponding to the destination subscriber's physical address. This method is simple and effective, but restrictive in that it requires subscribers to know each other's physical locations. Subscribers are not allowed to change networks or to have multiple network connections. [13]

Logical addressing assigns a permanent logical address to a subscriber which denotes one or more physical addresses associated with that subscriber. The sending subscriber does not need to know the physical location of the destination subscriber, and subscribers can relocate without change of address. [13]

Logical addressing of subscribers requires some form of "mapping table" for translation between logical and physical addresses. These tables must be stored at one or more locations in the network and updated when changes occur. The cost of maintaining these tables depends on the size of the network and the implementation of logical addressing selected. There are several possible locations for these tables, among those are: partial tables at each node, complete tables at each node, a distributed data base of tables at the nodes, or a centralized table at one or several locations. [16]

If there are only a few copies of the table scattered around the network, then nodes which do not contain the tables would have to query the nodes that do in order to perform the translation function. This is less efficient both in terms of overhead and response time than placing the table in every node. Considerations of reliability and survivability also favor placing the table in every node.

Several techniques for implementing logical addressing exist. Three of those techniques are presented here: complete mapping, partitioned mapping, and information service. [16]

In the complete mapping approach, all subscribers are logically addressed. Logical to physical address mapping is performed at the source node. The mapping table consists of N entries giving node number and port number, where N is the number of subscribers. When N is relatively small, the cost of the table in terms of storage required is insignificant. However, as N increases the costs increse exponentially.

In the partitioned mapping approach, the mapping table can be broken down into several tables corresponding to the parts of the address. In the hierarchical addressing scheme of the ARPAnet there would be three tables. One for the network, one for the switch, and one for the port.

The information service approach is based on the existence of one or more information service centers on the network. The center maintains the address mapping information for the network subscribers and provides it to the subscribers upon demand. This approach is most useful in large networks in which there are a few large central nodes and many smaller nodes with reduced capabilities. [13]

Additional Protocol Layer

An alternate approach to handling mobile subscribers would be to include an additional protocol at the transport level. This level would operate directly above the TCP layer. This scheme is based on separating the TCP connection identifiers from the physical addresses. Opening and closing TCP connections would be the responsibility of this new protocol in addition to maintaining data integrity as mobile subscribers moved out of one network and into another. The disadvantage of this approach is the necessity to transfer the mobile host's new address on a three way handshake basis, before the host moved onto the new network.

Summary

The next chapter will include further discussion of the two proposed solutions, logical and separate addressing, chosen to accomodate mobile subscribers in an internet environment. In this discussion, the two solutions are compared. Following this discussion the models used to simulate the environments are discussed.

III. Separate and Logical Addressing

When the TCP/IP protocols were designed, the requirement that mobile hosts must be handled was not included. Therefore, changes to either TCP or IP must be made to handle mobile subscribers and still keep TCP connections open while a subscriber moves from one network to another.

In this chapter, two solutions to the problem of addressing mobile subscribers in an internet environment are presented. The concept of logical and separate addressing is discussed. Next, the models used in analyzing these two solutions are presented.

Logical Addresses

Logical addressing allows the subscriber to change location without changing its logical address or name. This method of addresing allows the TCP connection to remain the same when the physical address changes. This occurs because the connection is based upon the logical address, not the physical address as is the case in the usual TCP/IP implementation.

The subscriber sends the name of the subscriber it wishes to communicate with to the TCP module. The TCP sets up the connection based upon its logical address. When the TCP sends out a message, it sends the source and destination logical addresses in the TCP header to the IP module located in its' node. The IP module handles mapping from logical to physical addresses. There is additional overhead in each segment as the logical and physical addresses are in the IP header. This method of passing addresses can be seen in figure 4. Whatever routing scheme is used is then based on this physical address. The logical address is carried along in the TCP header throughout the network so that mapping may take place whenever desired. For instance, mapping may take place only at the source and destination nodes or at every node between the source and destination nodes.

If the mapping takes place only at the source and destination nodes and the subscriber moves, then the subscribers' movement will not be discovered until it reaches the original destination. If mapping takes place more often, a segment may be rerouted before reaching the original destination, thus shortening delay. The method used in this study is that of mapping at each node.

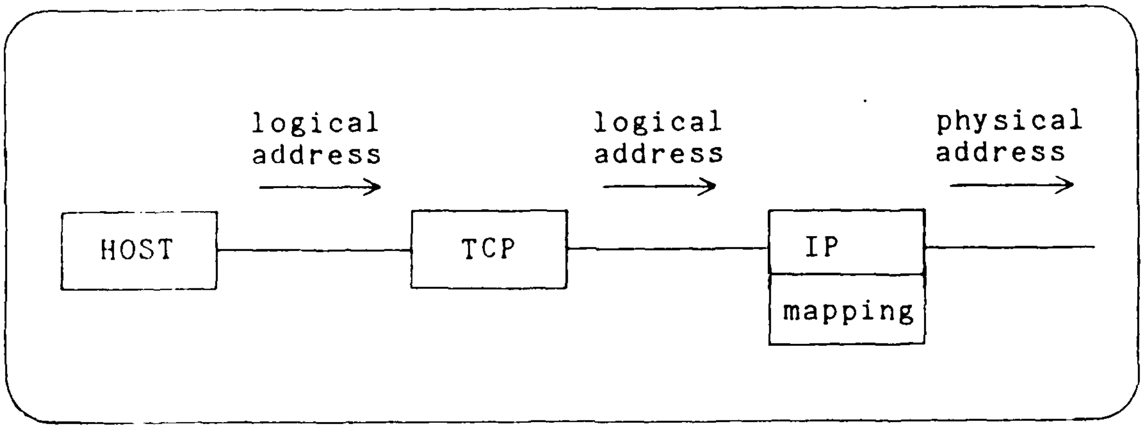


Figure 4. Logical Addressing

A possible problem with this addressing scheme is that of a segment racing around the network trying to catch up with a moving subscriber. This problem is more likely to occur when transmission rates are slow, the distance between locations is small, and the subscribers' vehicle is fast.

Another problem is that of updating the mapping tables. Many methods have been suggested in the literature on routing tables. In this study, the problem of updating tables has been simplified greatly. It is assumed that all tables are automatically updated by some central controller with no time delay.

Separate Addresses

Unbinding the TCP address from the IP address is what is meant by separate addresses. In other words, when a subscriber is mobile and changes networks, the TCP address remains static while the IP address changes to reflect the correct physical address.

The subscribei sends the name of the subscriber it wishes to communicte with to the TCP module. The TCP module maps this name and its host's name into the physical addresses of the destination and source subscribers and passes these physical addresses to the IP module in the node through the TCP header. The connection in this case would be based upon the names, once again, instead of the physical address. This method of passing addresses can be seen in figure 5.

The major disadvantage of this method of addressing is that when a subscriber moves during a data transmission, the move is not discovered until the segment reaches the original destination. Once the discovery is made, the data segment cannot be rerouted, but must be retransmitted with the new mapping. This is caused by the fact that the names are not passed along in the header as in the logical addressing method.

A problem which may be encountered in this method occurs when a subscriber moves during the establishment of a connection. When this happens the connection is not completed and the subscriber must start the connection procedure over.

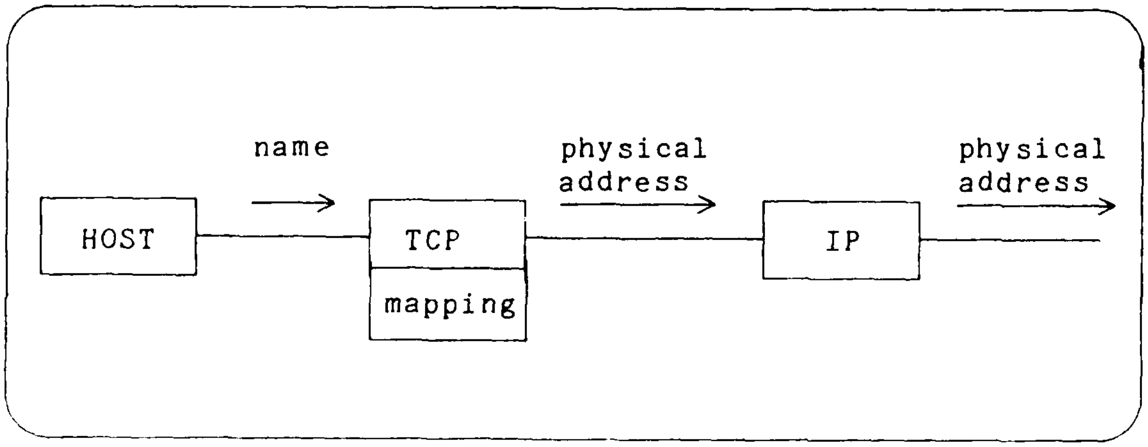


Figure 5. Separate Addressing

Differences

The main difference between these two forms of addressing is where the mapping from names to addresses takes place. In separate addressing, the mapping takes place in the host or the TCP module and in the logical addressing scheme, the mapping takes place in the node or the IP module.

Another difference is how a message is handled when a subscriber moves during transmission. In logical addressing, if the location of the destination changes, the message can be rerouted resulting in a savings of time in not having to retransmit the message caused by a timeout. In separate addressing, the only place the mapping can take place is at either end of the transmission. Therefore, if a message is in transmission and the destination host relocates, the message is lost and must be retransmitted. In either method, the connection is not broken as only the physical address changes, not the separate or logical address. In separate addressing, the connection must be reestablished if the subscriber moves during the connection process.

TCP/IP Queueing Models

The TCP functions of connections and data communications are represented as a network of queues in this simulation as depicted in figure 6. The connection procedures used by TCP opening and closing a connection can be seen in figure 7. A TCP connection is opened by TCP A sending a synchronize segment to TCP B. This is represented by the queues "connect request" and "connect accept" in figure 6. TCP B then sends a synchronize segment to acknowledge the synchronize segment it received from TCP A. This is represented by the queue "connect established". The connection is now established and data can be transmitted with the last acknowledge as can be seen in line 4 of figure 7. A TCP connection is closed by TCP A sending a finished segment to TCP B. This is represented by queues "close request" and "close wait". TCP B acknowledges the finished segment represented by queue "close wait-1". When TCP B is ready to close, it then sends a finished segment of its own to TCP A. This is represented by queues "close accept" and "close wait-2". TCP A sends an acknowledge back to TCP B represented by queues "close acknowledge" and "close delay". TCP A then delays for 2 MSL (maximum segment lifetime) which is represented by the queue "close delay". Using the descriptions of the connection management from [12] and [17], this function was decomposed into the 12 queues: connect request(OT1), connect accept(OR1), connect acknowledge(OT2), connect establish(OR2), close request(CT1), close wait(CR1), close wait-1(CT2), close accept(CR2), close wait-2(CT3), close acknowledge(CR3), close delay(CT4), and close delayr(CR4). A connection is initiated when a host starts the "connect request" queue. The queue then generates the segment which is passed to

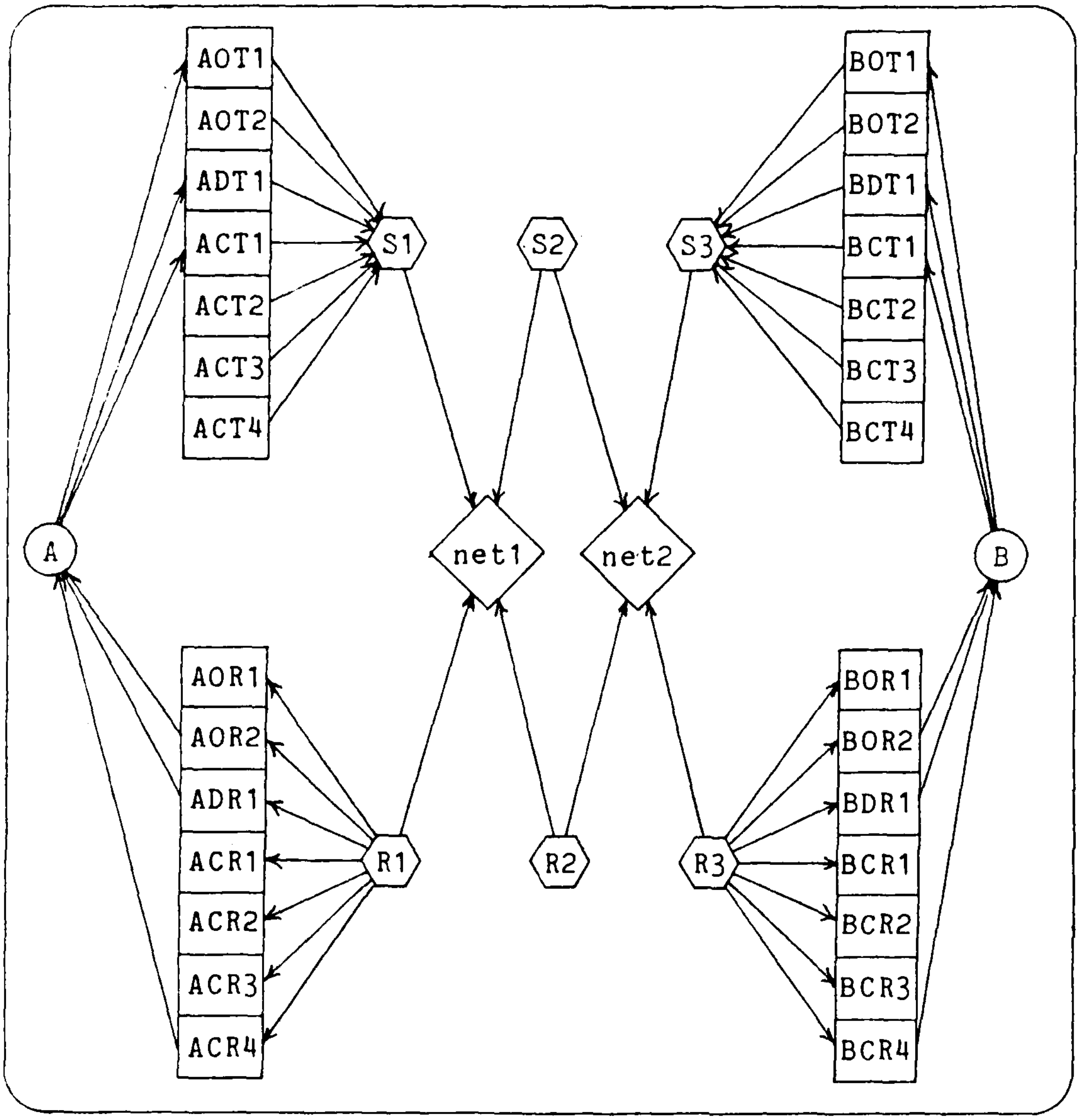


Figure 6. TCP/IP Queueing Model

the IP (IP send queue) in the node and to the channel where it is routed to the destination node (IP receive queue). At the destination node, it is assumed that the "connection accept" queue will always accept it. A further assumption is that only one connection may exist at any time. The "connection accept" queue in turn generates the second segment in the three-way handshake, which the first node always accepts at queue "connect establish". The third part of the handshake is passed through the data send queue(DT1) as described in [12:30]. The close connection process follows similarly through the queues "close request", "close wait", "close wait-1", "close accept", "close wait-2", "close acknowledge", "close delayt", and "close delayr". Data communications occurs through queues "data send" and "data receive"(DR1). These queues are all represented as M/M/1 queues.

```
    TCP A                              TCP B

1.  closed                               listen
2.  syn-sent      -------------->        syn-received
3.  established <--------------          syn-received
4.  established ----data----->           established

(a.) Basic 3-way handshake for connection synchronization


    TCP A                              TCP B

1.  established                          established
2.  fin-wait-1    ----------------->     close-wait
3.  fin-wait-2    <-----------------     close-wait
4.  time-wait     <-----------------     last-ack
5.  time-wait      ---------------->     closed
6.  closed

 (b.) close sequence
```

Figure 7. TCP Connection Management

Logical Addressing Model

The modification to the TCP/IP queueing models to obtain the model for logical addressing was simply to add a queue before the IP send queue. The function of this queue is the mapping of logical addresses to physical addresses. A queue is not needed before the IP receive queue as the logical address is carried in the TCP header allowing the IP module to examine the logical address to determine if at proper destination.

Instead of actually creating another queue, the service time of the send queues are increased to account for the mapping process. The mapping time is calculated in the same manner, based on lines of code, as is done for the other queues service time in the TCP/IP model. These calculations are explained in Chapter 4. The mapping process adds a delay to the IP send queue service time.

Separate Addressing Model

Modifications to the TCP/IP model to obtain the model for separate addressing are similar to the changes for logical addressing. The mapping of names to addresses takes place after each of the TCP transmit queues and before each TCP receive queue.

Again, as in the logical addressing model, the service times for the TCP queues are increased instead of creating new queues. In separate addressing, the names are not included in the TCP header, therefore mapping must take place at the receive queue to verify that the correct destination has been reached.

Summary

In this chapter, the two proposed solutions to the problem of addressing mobile subscribers are presented.

The basic TCP/IP model is examined and the changes made to this model to implement logical and separate addressing are presented.

The next chapter discusses the simulation programs created to analyze the performance characteristics of the addressing methods presented in this chapter.

The basic TCP/IP model is examined and the changes made to this model to implement logical and separate addressing are presented.

The next chapter discusses the simulation programs created to analyze the performance characteristics of the addressing methods presented in this chapter.

IV. Simulation

The objective of the simulation is to test the proposed modifications to the TCP/IP model to allow mobile subscribers by selecting and fixing some network parameters, and then making multiple runs in which the remaining parameters are varied. Though limited in scope, the simulations provide performance characteristics for comparison. These performance characteristics are data segment delay and throughput. The simulations and their parameters are the topic of this chapter.

Test Network

The test network simulated is shown in figure 10. The system of networks simulated was chosen so as to represent the typical environment for which TCP/IP were designed. This typical environment has been described as a "catenet" [12:1] or an ARPA model [10]. In this environment, hosts are connected to a number of networks. Each host, therefore, has access to any other host through an arbitrary path, comprised of a series of networks and gateways, which is determined by the lower level protocols.

In the test network there are three nodes and two subscribers. One subscriber, designated as host-2 in figure 8, is the mobile subscriber in the simulation of the logical and separate addressing models.

This simulation assumes that a datagram must pass through four gateways on its path between nodes. These

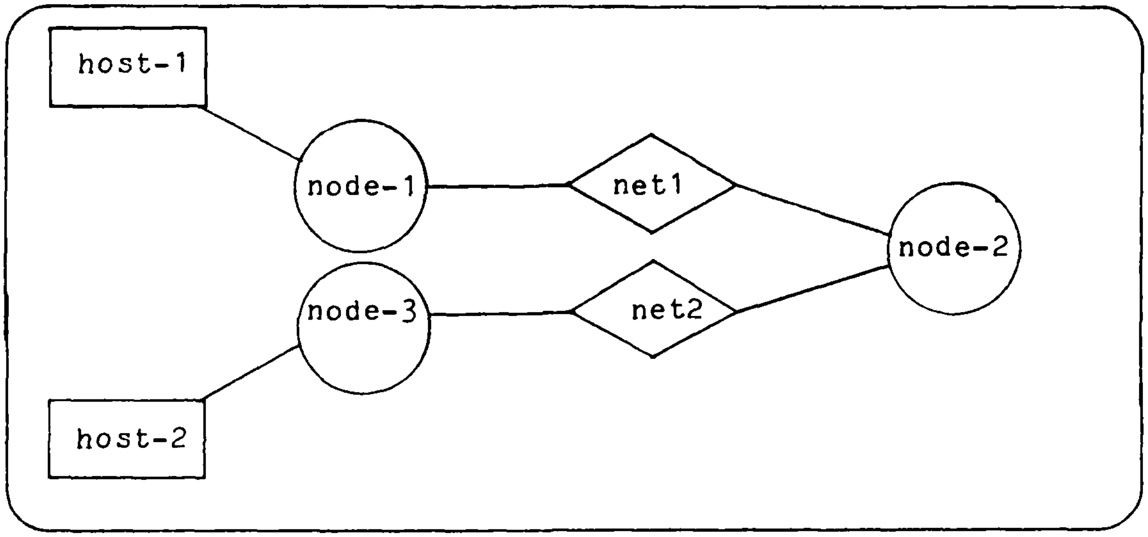
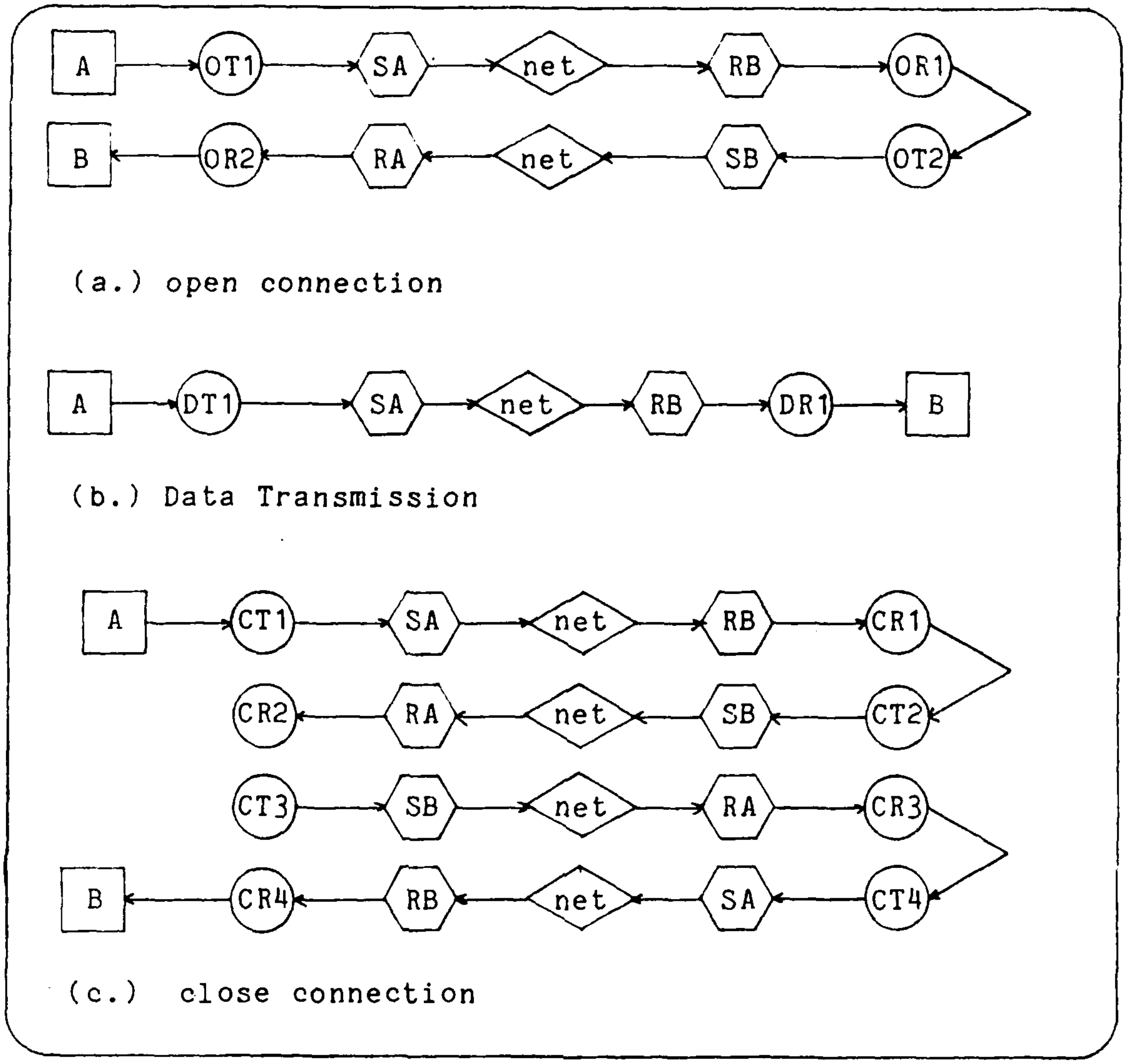


Figure 8. The Simulation Network

gateways are designated as net1 and net2 in figure 8. Each gateway performs the functions of interpreting the incoming IP header and building a new IP header for the outgoing datagram.

Simulation Characteristics

The path which a message may follow is called a chain [19:206]. In this simulation, there are three possible chains for a message to follow. The first of these three chains occurs only when a connection is opened between two subscribers. The second chain, which is the path that a data message follows, is repeated until the third chain is activated. The third chain is that of closing the connection. The paths followed by all three chains can be seen in figure 9.



(a.) open connection

(b.) Data Transmission

(c.) close connection

Figure 9. Queueing Chains

Simulation parameters

In view of the wide variety of parameters that could have been varied in these simulations, many were fixed at what was considered to be reasonable approximations. Parameters varied include data packet size, number of octets in the headers, delay due to mapping addresses, and the mobility of subscribers. Discussed first are the fixed parameters.

Arrival rates were selected aroitrarily while the service rates were estimated using the partially implemented TCP/IP network described in [18]. From the PLZ program listings given in [18] and the estimate that one line of PLZ executing on a Z80 based system takes approximately 160 msec (200 nsec Z80 cycle time takes 10 cycles per Z80 instruction (approx.) times 8 Z80 instructions per PLZ line (approx.)) the minimum time required to build a TCP segment (header and data) was estimated to be 3.84E-3 seconds. This minimum was used as the average service rate for each of the TCP transmit queues (OT1, OT2, DT1, CT1, CT2, CT3, and CT4 as shown in figure 6). However, if any service rate less than the minimum was generated by the exponential distribution function, then the average service rate was used instead. It was felt that this better represented the effects of other processes competing for the host's or nodes resources. Similarly, the time to extract and interpret the TCP header, the time to build an IP header and the time to extract and interpret an IP header were found to be 3.252E-3 seconds (queues OR1, OR2, DR1, CR1, CR2, CR3, and CR4), 6.56E-3 seconds (IP send), and 6.24E-3 seconds (IP receive), respectively. These times were also used as minimums.

The time between subscribers changing locations was calculated using an exponential distribution with an average time between locations to be 8.5 minutes. This average time was based on a vehicle speed of 210 km/hr. The assumption here is that the vehicle is moving continuously.

The type of traffic simulated is interactive with varying arrival rates. What is meant by interactive traffic is that the subscribers exhange data segments. The traffic was generated by exponentially varying the data size that the segments were to carry. Error rates were assumed to be zero. In other words, a segmen that is transmitted will be received free of errors and will not have to be retransmitted. As seen in Chapter 1, the maximum length of a TCP header is 24 octets and the maximum length of the IP header is 68 octets. Therefore, the maximum number of overhead octets allowed in a TCP/IP segment header is 92 for the baseline version and the separate addressing version. There are an additional 8 octets for the source and destination names in the logical addressing version. Once a connection is established, the interarrival time intervals of interactive traffic to the system is assumed exponentially distributed. Connections and the durations of the connections are also established and maintained assuming exponential distributions.

As mentioned in Chapter 2, the maximum datagram length allowed by the IP is 64K octets. All hosts must be prepared to accept datagrams of up to 576 octets whether they arrive whole or in fragments [12:13]. The average length of a datagram used in this simulation is varied between 50 and 500 octets.

Other parameters to be selected include the distances between nodes, the propogation delay in the networks and the data rate of the transmission medium. The distance between gateways was chosen to be 30km. The average time to transmit one bit over a one kilometer link is 6 microseconds per kilometer. The propogation delay per bit is calculated to be the distance between gateways multiplied by the average time to transmit one bit. In these simulations the propogation delay per bit is .0018 sec (30km * 6E-6 sec/km) [11]. The transmit time between nodes is found by multiplying the inverse of the data rate by the total number of bits in the datagram. The data rate is 1.544Mbps, which is the CCITT standard gross data rate of a channel. The total delay between nodes is then the propogation delay of the first bit plus the transmission time.

Programming Scheme

This section discusses the simulation program which is a modified version of code found in reference 20. The program organization is looked at along with the functions of some of the modules. The simulation program was written in the Pascal language. The simulation program was organized as a set of procedures and functions controlled by a simulation clock (figure 10). Before the simulation begins, the routine is setup by the main program calling a procedure named "initialize". "Initialize" reads input variables, sets up routes, initializes queue and data segment statistics, and prints out various input parameters.

The main program next calls the procedure "start". The procedure "start" initializes the open connection and close connection paths. An event is inserted in the event list for the connect request transmit queue at time current clock + time between connections * ln(random #). An event is also inserted in the event list for the close request transmit queue at time for connection to begin + length of connection * ln(random #). The system state is set to connect request.

The simulation is now ready to begin. The main program calls the procedure "loop" which runs through a number of connections. The basic algorithm is for an event to arrive at a queue, complete service at a queue, and then determine which queue to arrive at next. In this simulation an event can be thought of as a data segment moving through the network.

After the simulation has run a number of connections, the main program calls the procedure "printstats". "Printstats" uses the statistics collected to calculate throughput and delay, then prints out the results.

The procedure "removefirstevent" removes an event from the event list. "Removefirstevent" has a clock called cumclock which is the amount of time segments are in the

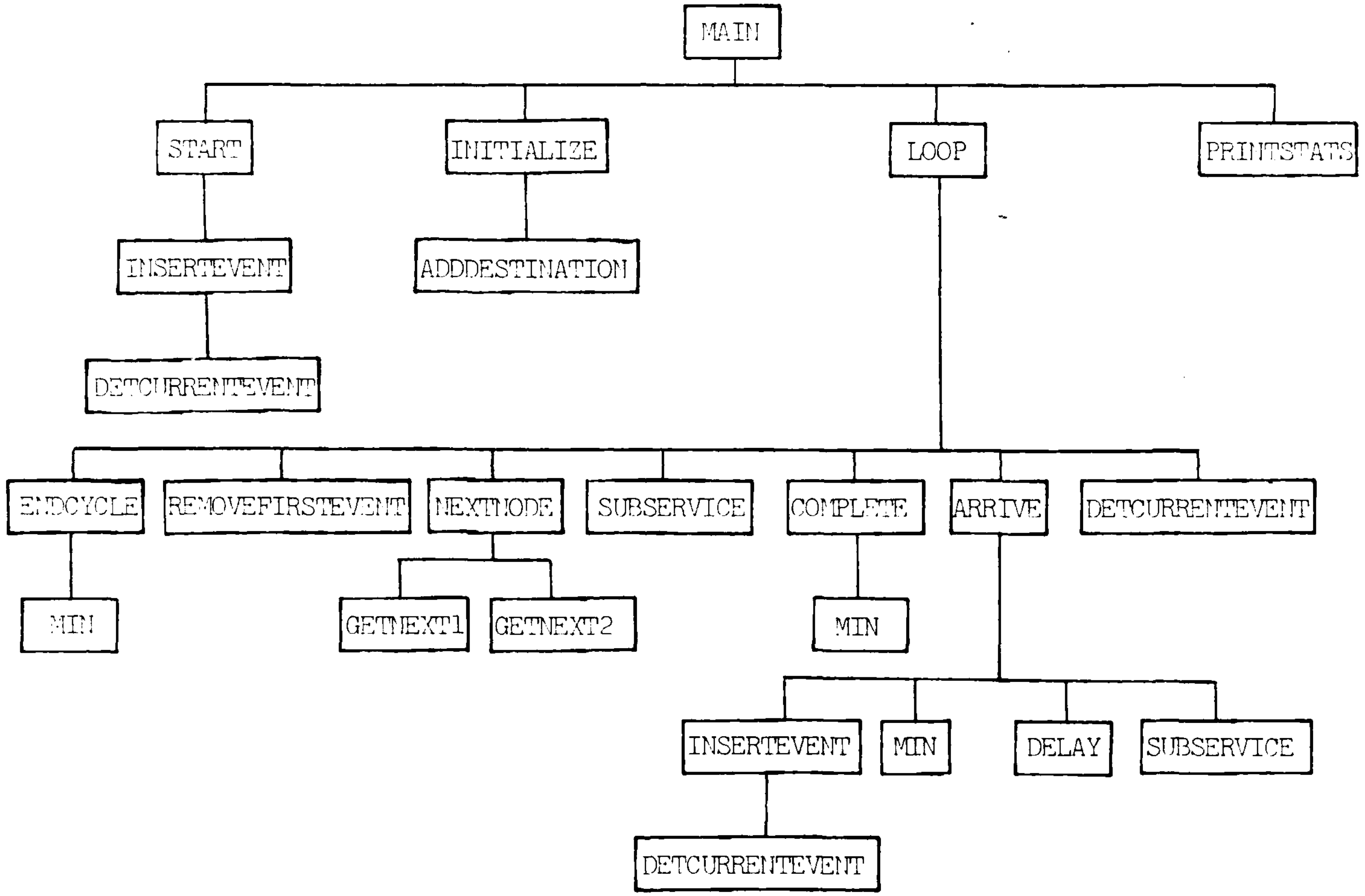


Figure 10. Program Hierarchy

system. The datacumclock is the amount of time data segments are in the system, which is also updated by this procedure.

The procedure "complete" updates statistics for the queue just removed from the event list by procedure "removefirstevent".

The procedure "arrive" updates statistics for the queue determined by the procedure nextnode. "Arrive" inserts an event in the event list to be removed when service is completed. The procedure "nextnode" determines which node is next in the route from one subscriber to the next. This procedure is different for logical and separate addressing. The changes to the baseline code described here is found in the following sections. In the baseline model the routing decisions are few, as the subscribers are nonmobile. The only queues where a decision must be made are at the IP receive queues. These decisions are based on the system state. For example, if the state of the system is connectreq, the receiving node would be OR1; if the state were closereq, the receiving node would be CR1.

The structure charts and data dictionary for the baseline model can be seen in Appendix A. The program code can be found in Appendix B.

Program Modifications for Logical Addressing

The changes to the baseline code for the logical addressing scheme can be seen in Appendix C. A new data record called subscriber has been created to keep track of where each subscriber is located. Information relative to the receiving subscriber location is vital to the procedure "nextnode" in determining where to send the segment in transit.

Program Modifications for Separate Addressing

The changes to the baseline code for the separate addressing scheme can be seen in Appendix D. The data record subscriber, used in the logical addressing scheme, is also used here. Added to this record are the subscribers' new location after moving along with a flag called movedflag indicating that the subscriber has moved during a transmission. These new items are needed due to the fact that a segment cannot be rerouted during transmission. This fact is reflected in the code found in procedure "nextnode" by the nextnode being set to data transmit, DT1, when the data segment being sent reaches the original destination and finds that the subscriber is no longer there. If a subscriber changes location while a connection is being opened, the connection is broken and must be reinitiated. The same connection is not automatically reestablished; the originating subscriber must request a connection with the destination subscriber again. This is done in the program by the event list being set to empty, thus ending the simulation of that connection. In this version, decision points for routing occur at the node-2 IP send queue in addition to each IP receive queue. A decision point occurs at the node-2 IP send queue because the receiving subscriber may be in either net1 or net2. See figure 10 for network setup.

Summary

The results of the simulation along with the analysis of the results are found in Chapter 5. The effective throughput and data segment delay are compared for data segments of varying size.

V. Performance Analysis of Addressing Schemes

The results discussed in this chapter involve the analysis of simulations of the baseline, logical addressing, and separate addressing models. The analysis centers around several parameters which were considered to have the broadest potential effect on the system throughput and delay. The results of the simulation runs in terms of throughput and delay are summarized in Tables I and II and examined in the following sections. The paramters analyzed are compared for various segment data and header sizes and when subscriber-2 is either mobile or stationary. The different models are examined when the subscribers are stationary to determine the amount of performance degradation to the network when subscribers are mobile. First the impact on the throughput performance is analyzed, then the degradation in delay is examined.

Throughput

The throughput calculated is actually an effective throughput, that is, based on data only. The throughput is calculated as the total number of data bits divided by the total time that segments are in the system. For each simulation run, the data size was varied from 50 to 500 octets.

Nonmobile. This section presents the results of the simulation when the subscribers are not mobile. In order to compare nonmobile with mobile results later, the simulation is run three times. In each run subscriber-1 is attached to node-2. Subscriber-2 is attached to each of the three nodes. The results are then averaged over the three runs to obtain results which can be compared with the mobile runs. Each of the three models were simulated with the TCP/IP header size varying from minimum to maximum and the data size varying from 50 octets to 500 octets.

In the baseline model the simulation is run with header sizes of 20, 60 and 92 octets. The values of 20 and 92 are the minimum and maximum values allowed by the TCP/IP protocols and 60 octets is a commonly used header size. For the three data segment sizes used, there is very little change seen in the throughput. Generally, there is a slight downward trend in throughput as the header size increases, with a large upward trend in throughput when data size increases. The throughput decreases by less than .5% for the larg header size. Because of this small difference due to header size, the remaining discussion will be based on the average header size. The throughput increases as the size increases mainly due to the channel use increasing. These results are plotted in figure 11.

In the logical addressing model the simulation is run with header sizes at 28, 68, and 100 octets. These values are obtained by adding the 8 octets for address mapping to the minimum, average, and maximum values used in the baseline simulation. As in the baseline simulation, the throughput when the header size is at maximum size is 99.8% of the throughput for minimum header size.

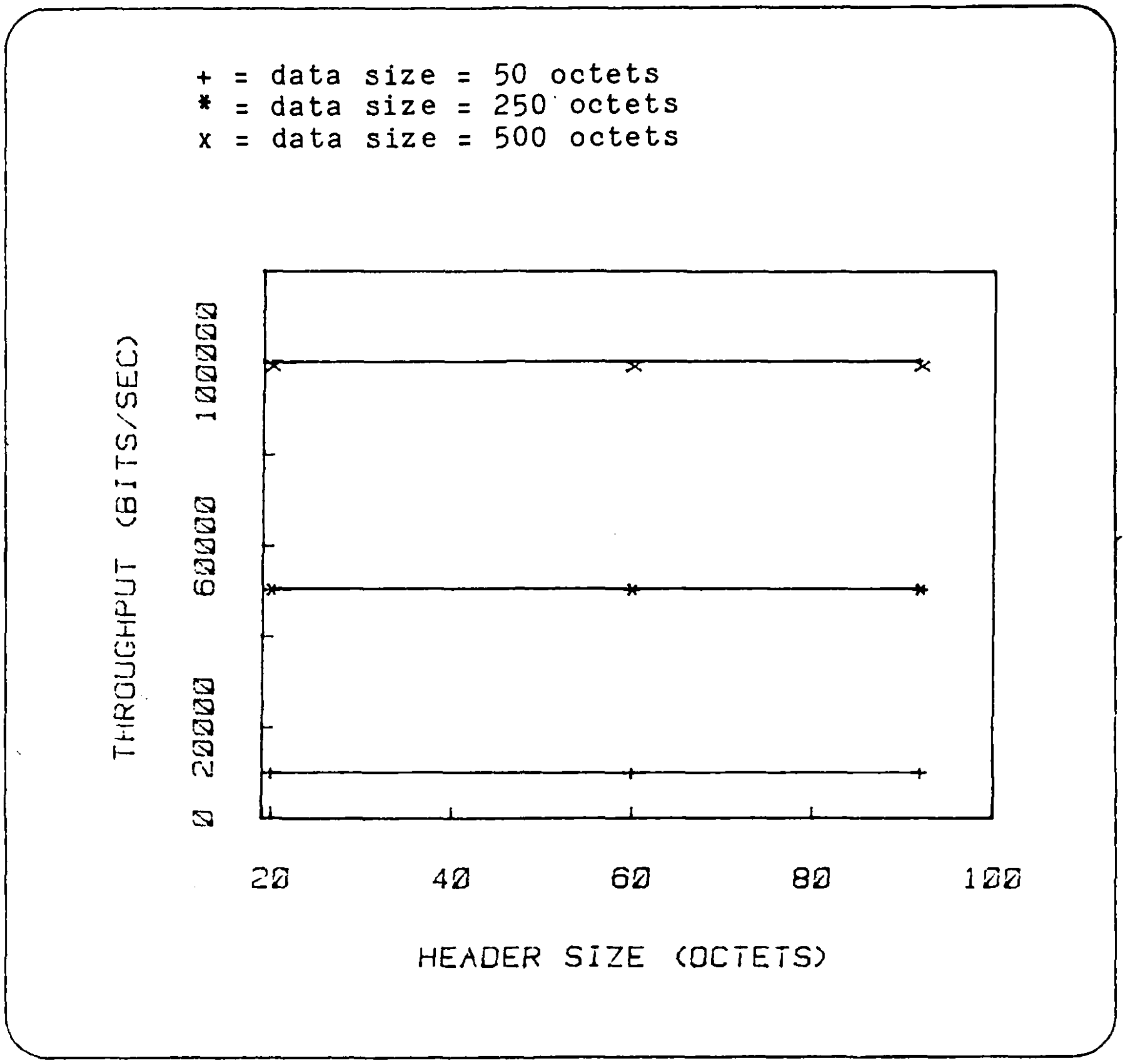


Figure 11. Baseline Model, Nonmobile Throughput

In the separate addressing model, the simulation is run with header sizes at 20, 60 and 92 octets. As in both the previous models the throughput for maximum header size and the throughput for minimum header size are almost the same with only .1% difference.

In comparing the three models, the trend is similar for all three header sizes. The throughput for the logical

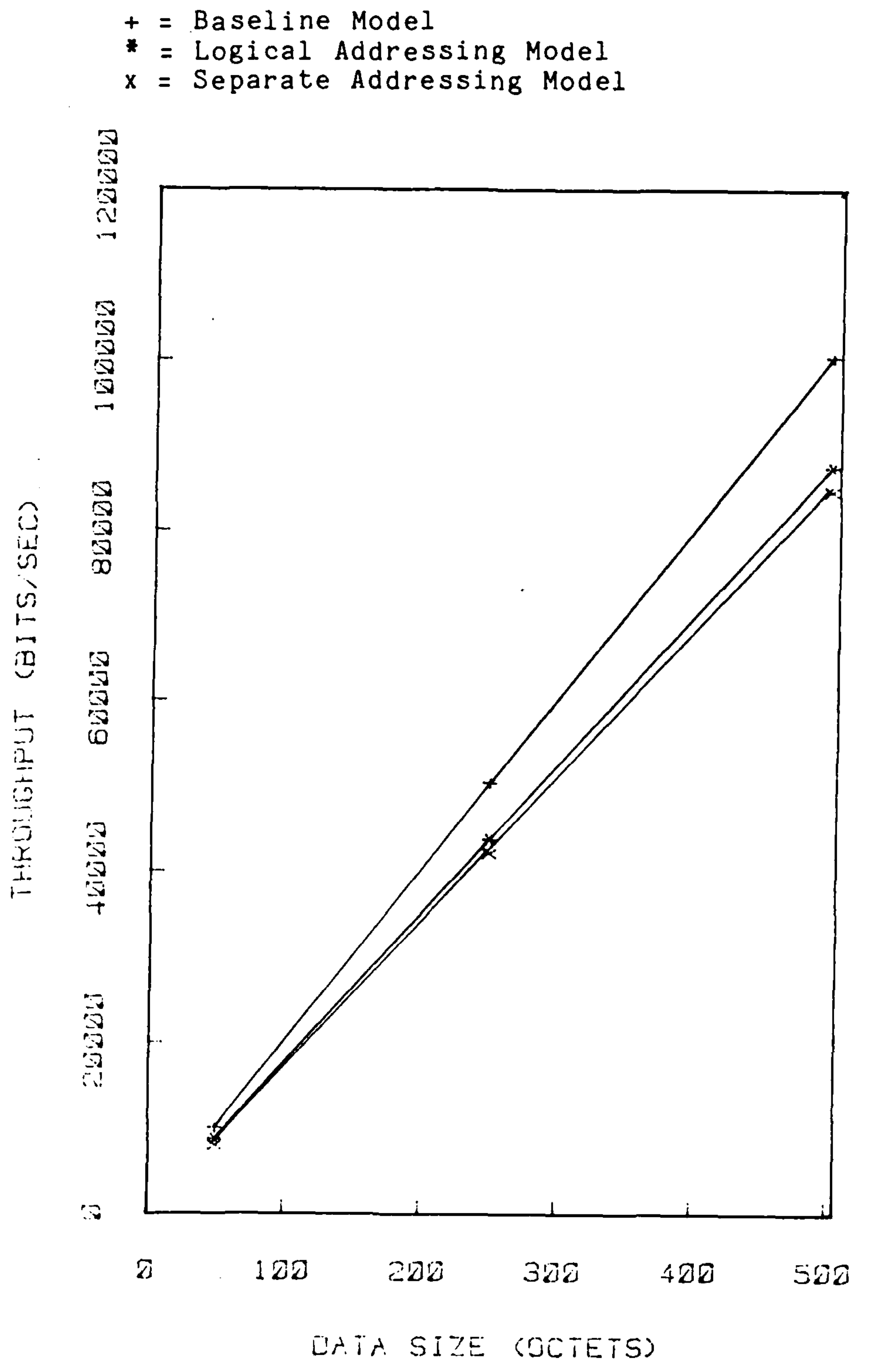


Figure 12. Nonmobile Subscribers, Throughput Summary

addressing model is 85.3% of the throughput for the baseline model and 88.26% of the throughput for the separate addressing model. Throughput for the separate addressing model is 96.9% of the throughput for the baseline model. This can be seen in figure 12.

Mobile. This section compares the results of the simulation when the subscribers are allowed to move around the test network system. Subscriber-1, the nonmobile subscriber, is located at node-2. Subscriber-2, the mobile subscriber, starts at node-3. Subscriber-2 is allowed to move through the network system with an average time between changing location of 8.5 minutes. The mobile subscriber travels through the network system in a circular pattern. The mobile subscriber rotates through node-3, node-1, node-2, and then back to node-3 again. The results of varying the TCP/IP header size for the logical and separate addressing models are presented.

As is the case of nonmobile subscribers, the throughput for maximum header size is slightly less than the throughput for minimum header size. There is a .5% difference in throughput for the logical addressing model and a .6% difference in the throughput for the separate addressing model. The throughput for logical addressing is 93.5% of the throughput for separate addressing. The data for these runs is plotted in figure 13.

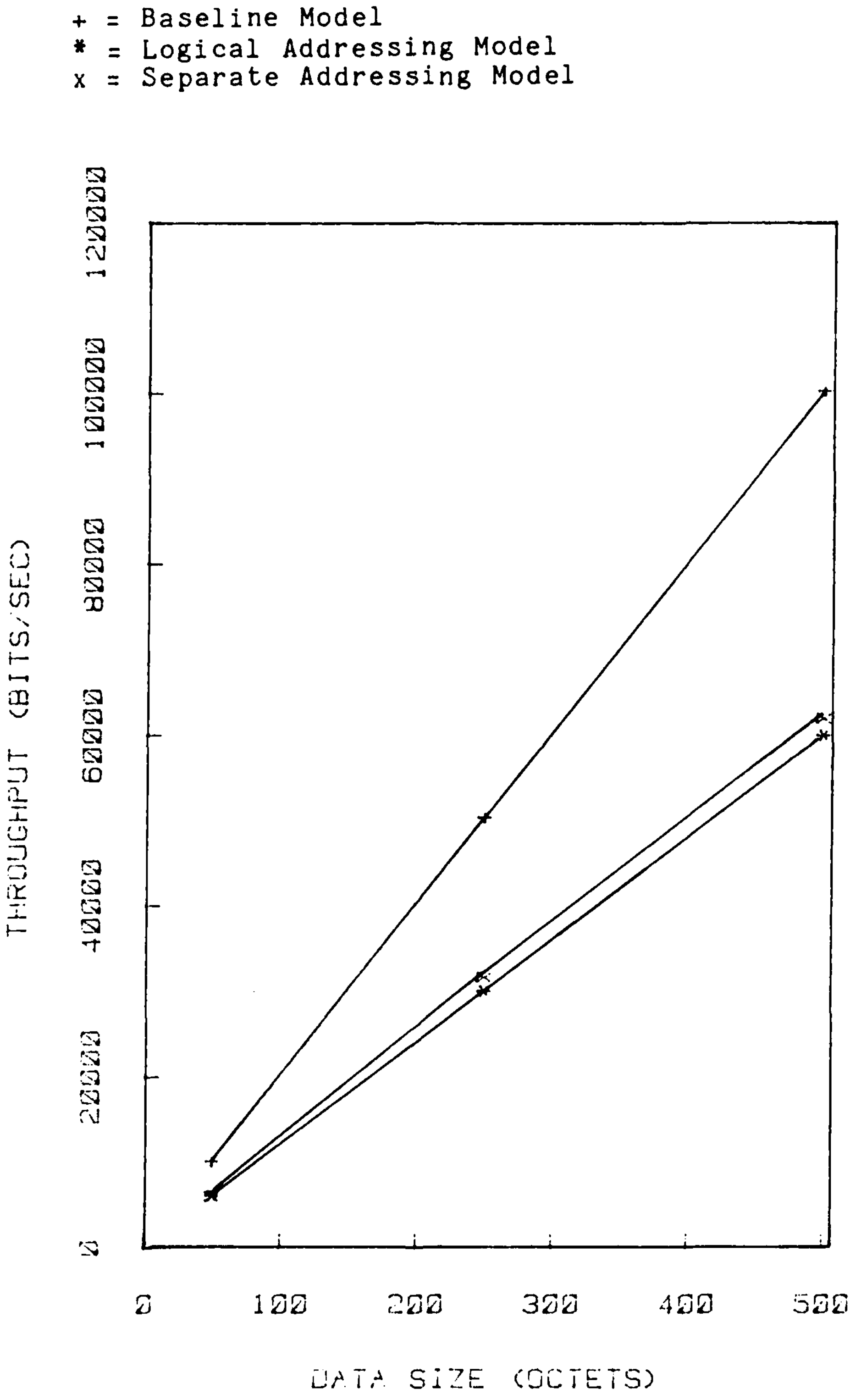


Figure 13. Mobile Subscribers, Throughput Summary

Nonmobile versus mobile. When comparing mobile to nonmobile in similar network environments, the mobile throughput is 70.6% of the nonmobile throughput for logical addressing and the mobile throughput is 81.3% of the nonmobile throughput for separate addressing. Overall, the throughput goes down when mobile subscribers are present. The throughput for the separate addressing model is larger than for logical addressing. The results are plotted in figure 14.

Delay

The delay is actually the average data segment delay. Delay is calculated by dividing by the "datacumclock" (total time data segments are in the system) by the total number of data segments. For each simulation run, the data size is varied from 50 to 500 octets for header sizes from minimum to maximum.

Nonmobile. This section presents the results of the simulation when the subscribers are not allowed to move. Each of the three models are simulated with the TCP/IP header size varying from minimum to maximum allowed.

In the baseline model the minimum header size creates a delay which is 99.6% of the delay created when the header size is at maximum allowed. The results are the same for the logical and separate addressing models. In general, the delay increases as the header size increases and as the data size increases, thus finding the largest delay when

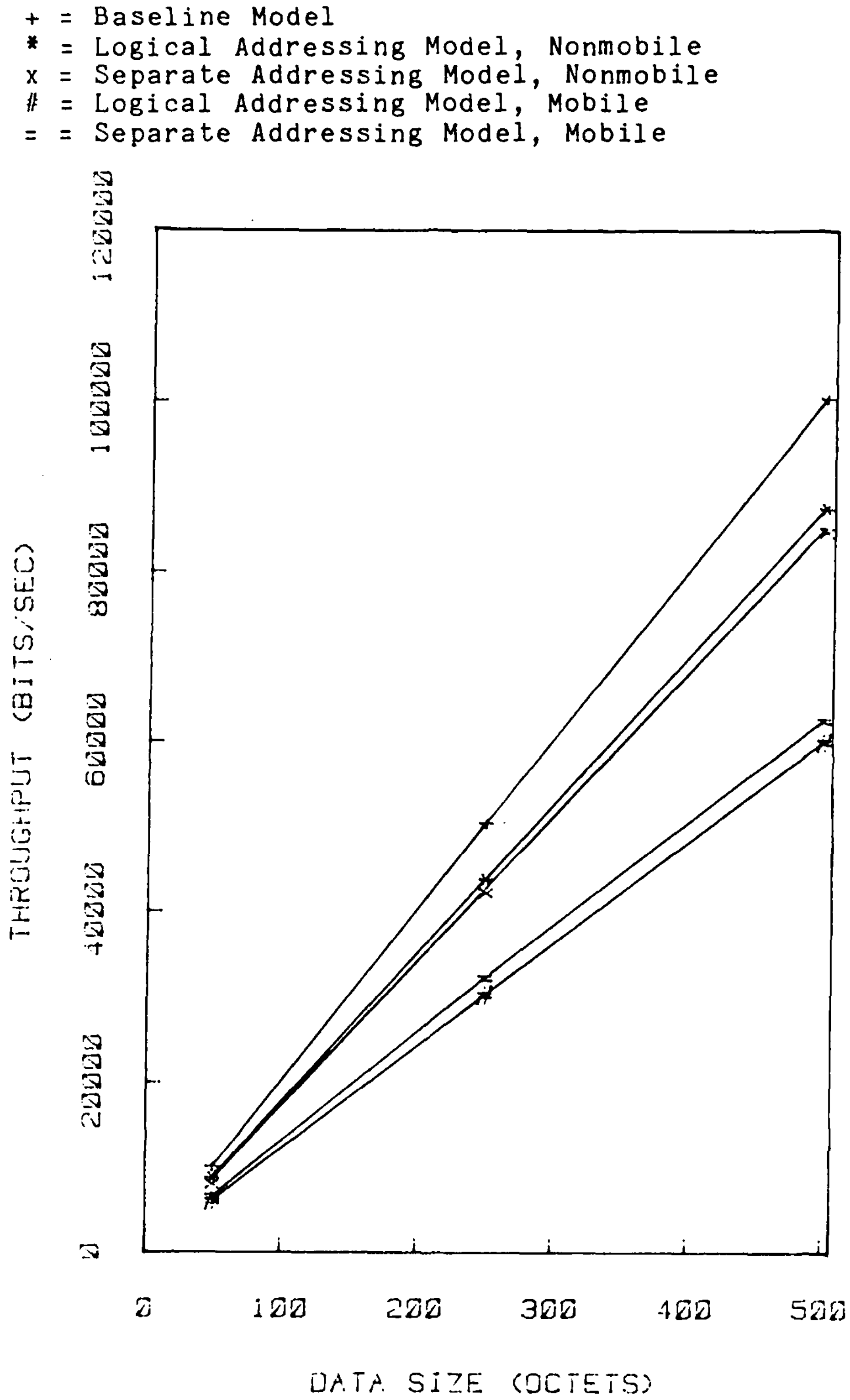


Figure 14. Throughput Summary

the data size is 500 octets and the header size is 92 or 100 octets.

In comparing the delays between the three models, it can be seen that the lowest delay occurs in the baseline model. The delay is lower in the logical addressing version than in the separate addressing version. The average delay for the baseline model is 82.8% of the delay for the separate addressing model and 86% of the delay for the logical addressing model. These results are shown in figure 15.

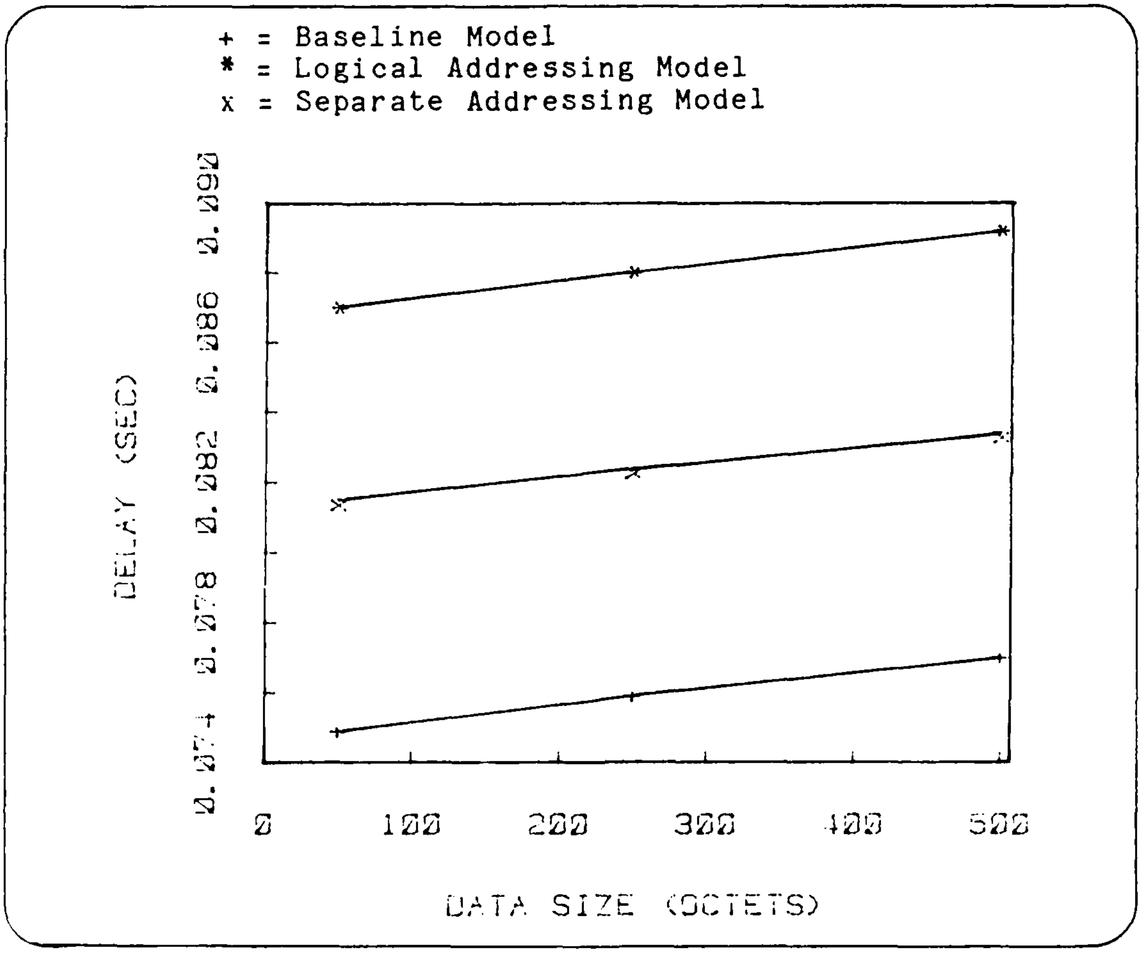


Figure 15. Nonmobile Subscribers, Delay Summary

Mobile. This section presents the results of the simulation when a subscriber is allowed to move through the system. The logical and separate addressing models were run with the TCP/IP header size varying from minimum to maximum allowed.

As was the case for nonmobile, the delay for the segments with a minimum header size is 99.6% of the delay for the segments with maximum header size.

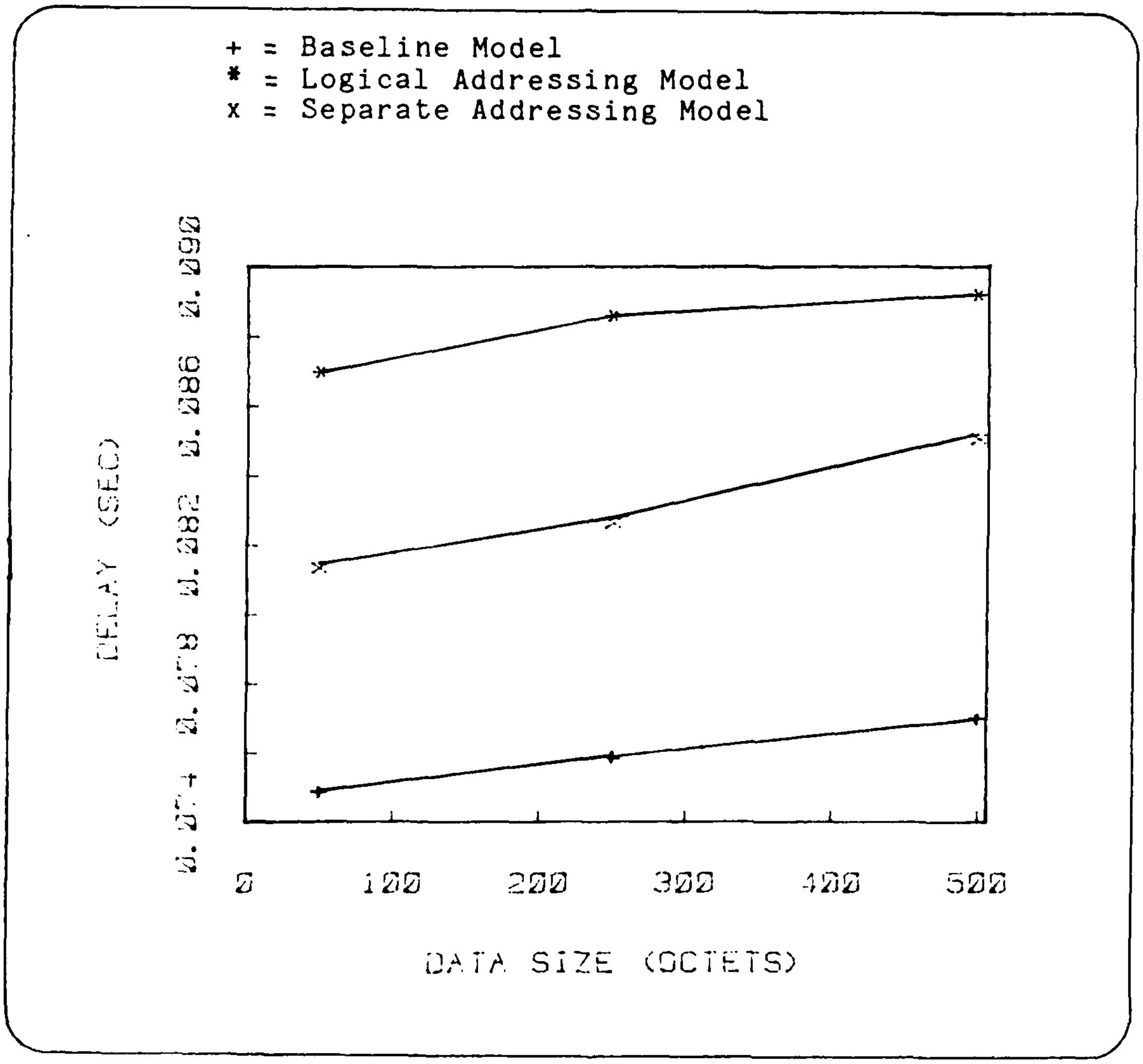


Figure 16. Mobile Subscribers, Delay Summary

In comparing the two models with the baseline version, which can be seen in figure 16, the delay for the separate addressing model is now lower than the :ogical addressing model. Delays are still longer for logical and separat^ addressing than for the baseline model.

Nonmobile versus mobile. In both the logical and separate addressing models, the delay increases in the mobile runs. In the logical addressing model the average nonmobile delay is 98.2% of the mobile delay and the average nonmobile delay is 90.1% of the mobile delay in the separate addressing model. These results are plotted in figure 17.

The differences between the mobile and nonmobile delays appear to change depending upon data size. The only explanation for this would be the effect the simulation itself has on the results. For instance, the number of times a subscriber moves during transmissions will affect the delay.

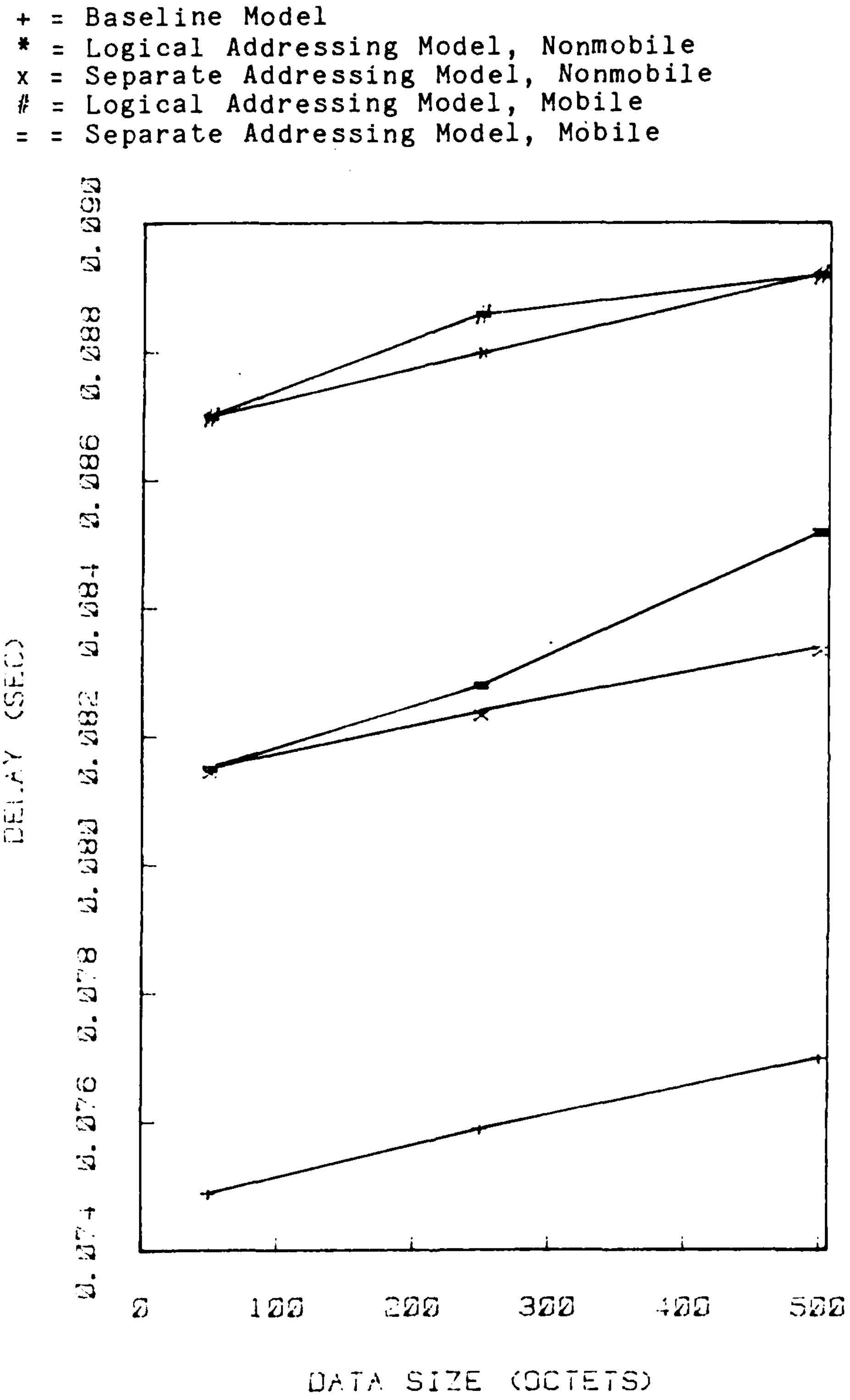


Figure 17. Delay Summary

Table I

Throughput (bits/sec)

Model	Data Size (octets)	Min. Header Size (20 octets)	Ave. Header Size (60 octets)	Max. Header Size (92 octets)
		Nonmobile		
Base	50	10,153	10,132	10,122
	250	50,427	50,390	50,410
	500	100,445	100,380	100,388
Logical	50	8,838	8,829	8,824
	250	43,856	43,938	43,922
	500	87,650	87,491	87,449
Separate	50	8,206	8,854	8,575
	250	41,008	42,727	43,922
	500	81,457	85,224	85,100
		Mobile		
Logical	50	6,191	6,131	6,177
	250	31,349	30,134	30,138
	500	60,149	60,071	60,283
Separate	50	6,828	6,561	6,511
	250	32,609	32,252	32,087
	500	63,345	62,553	63,588

Table II

Delay (sec)

Model	Data Size (Octets)	Min. Header Size (20 octets)	Ave. Header Size (60 octets)	Max. Header Size (92 octets)
		Nonmobile		
Base	50	.0747	.0749	.0749
	250	.0757	.0759	.0759
	500	.0769	.0770	.0771
Logical	50	.0868	.0870	.0872
	250	.0880	.0880	.0882
	500	.0891	.0892	.0894
Separate	50	.0902	.0815	.0817
	250	.0912	.0824	.0825
	500	.0928	.0834	.0837
		Mobile		
Logical	50	.0861	.0870	.0863
	250	.0857	.0886	.0888
	500	.0891	.0892	.0888
Separate	50	.0785	.0815	.0820
	250	.0819	.0828	.0831
	500	.0843	.0852	.0839

Summary

To summarize the results, figure 14 shows the overall trends for throughput and figure 17 shows the overall trends for delay. Figure 14 shows that throughput decreases when logical or separate addressing is implemented with the larger decrease resulting with logical addressing. Figure 17 shows the delay is increased when either scheme is implemented with a larger delay in the separate addressing scheme.

Chapter 6 contains conclusions which may be drawn from the results presented here. Also in Chapter 6 are recommendations for further study.

VI. Conclusion and Recommendations

This chapter contains the conclusions which are based on the performance analysis found in Chapter 5. Also contained in this chapter are suggestions for further study in this area.

Conclusions

As was seen in Chapter 5, there is a price to pay for mobility. This price is paid in terms of decreased throughput and increased delay for data. There are other costs such as more memory at each node for the logical addressing model and at each subscriber for the separate addressing model for the address mapping function. The impact the need for additional memory for address mapping has on the alternate schemes would depend on the size of the network system and the number of subscribers attached to the system. If there are amny nodes and only a few subscribers, the logical addressing scheme would require a greater amount of additional memory. When the network system is small, i.e. very few nodes, and many subscribers are attached to those nodes, the separate addressing scheme requires a greater amount of additional memory for the address mapping function. Another cost would be that of a central controller, or some other scheme to monitor the movement of mobile subscribers and to update the mapping tables. The cost of a central controller would be a common cost to any addressing scheme, although the actual updating of mapping tables will vary according to the scheme. These other costs will not be further discussed here but are nevertheless an important consideration in the implementation of an alternate addressing scheme.

Indicators which are used for comparison are throughput and delay. As was seen in Chapter 5, throughput decreases more for the separate addressing model than for the logical addressing model while the delay has a greater increase for the logical addressing model that for the separate addressing model. The possible causes for these increases and decreases and the differences between the models will be looked at next.

The most probable reason that the throughput decreases in the separate addressing model is the fact that when a subscriber moves during transmission of a data segment; that segment must be retransmitted. In the logical addressing model, the data segment is not retransmitted, but rather rerouted, which would add a delay, but not as signficant as in the separate addressing scheme. The biggest cause for decreased throughput is the increased time at each node due to mapping of addresses.

The increased delay in both alternate addressing schemes is due to the mapping of addresses. The reason for the longer delays in the logical addressing model is due to the fact that the mapping occurs at each node. In the separate addressing model the mapping occurs only at each end of the transmission.

In choosing which method of addressing is more cost-effective in terms of throughput and delay, the choice would be separate addressing. Although the throughput is lower when the subscribers are not mobile, the throughput is greater and the delay is lower when the subscribers are mobile.

Recommendations for Further Study

There were many simplifying assumptions made to simulate this test network, one of which was the test network itself. Expanding the test network to contain more that two connected networks would require a more sophisticated routing scheme. One approach to this problem would be to use a routing table. This routing table would find a route for a data segment to use based on the nodes that the subscribers are attached to. Allowing more than one mobile subscriber would raise the problem of updating the mapping table entries simultaneously. In allowing both subscribers to move, it would be interesting to compare the number of retransmissions required, the number of connections to be reestablished in addition to any changes in throughput and delay.

Analyzing the effect of errors in transmission in both the nonmobile and mobile environment is another area to be investigated. Errors in transmission could be generated by exponentially varying the number of good data segments transmitted before an error occurs.

The mapping process and its management is in itself an entire area to be studied. There could be a central controller or each subscriber or node could be responsible for updating the mapping tables. Another problem is how the manager obtains the address of each subscriber.

APPENDIX A: Data Dictionary and Structure Charts

This appendix contains the data dictionary and structure charts for the simulation programs used to analyze the performance characteristics of mobile subscribers in a multi-network environment. Included in the data dictionary are entries for all variables and modules.

Data Dictionary

Variables

VARIABLE: arrivalflag
TYPE: boolean
ALIASES: none
USED: insertevent, arrive
FUNCTION: indicates to the procedure insertevent that the event is from arrive

VARIABLE: avail
TYPE: elemptr
ALIASES: none
USED: insertevent, removefirstevent, initialize
FUNCTION: pointer to element record

VARIABLE: availrouting
TYPE: routingptr
ALIASES: none
USED: adddestination, initialize
FUNCTION: pointer of routingelement record

VARIABLE: avedatasegmentdelay
TYPE: real
ALIASES: none
USED: printstats
FUNCTION: how long it takes a data segment to propogate through the system

VARIABLE: avemovetime
TYPE: real
ALIASES: none
USED: start, initialize, loop
FUNCTION: average amount of time a mobile subscriber remains connected to a node

VARIABLE: avesegmentdelay
TYPE: real
ALIASES: none
USED: printstats
FUNCTION: how long it takes a segment to propogate through the system

VARIABLE: clock
TYPE: real
ALIASES: none
USED: endcycle, insertevent, removefirstevent, start, arrive, complete, initialize, loop, printstats
FUNCTION: used to keep track of time in the simulation

VARIABLE: closeq
TYPE: integer
ALIASES: none
USED: loop
FUNCTION: indicates what queue is waiting on a data data segment to be received.

VARIABLE: cumclock
TYPE: real
ALIASES: none
USED: removefirstevent, printstats, initialize
FUNCTION: time spent transmitting segments

VARIABLE: currentdataoctets
TYPE: integer
ALIASES: none
USED: delay, loop, arrive, start, removefirstevent, insertevent, initialize
FUNCTION: number of data octets in this message

VARIABLE: currentevent
TYPE: state
ALIASES: none
USED: nextnode, loop, start, detcurrentevent, getnext1, getnext2
FUNCTION: state of the simulation

VARIABLE: cycles
TYPE: integer
ALIASES: none
USED: endcycle, start, printstats, initiailize
FUNCTION: number of connections that have been completed

VARIABLE: dataarrival
TYPE: real
ALIASES: none
USED: initialize, loop
FUNCTION: average time between data segments

VARIABLE: dataclock
TYPE: real
ALIASES: none
USED: loop
FUNCTION: time a data event is due to arrive

VARIABLE: datacumclock
TYPE: real
ALIASES: none
USED: removefirstevent, printstats, initialize
FUNCTION: amount of time during simulation that data segments were transmitted

VARIABLE: dataretransmit
TYPE: integer
ALIASES: none
USED: nextnode, printstats, initialize
FUNCTION: number of times data segments had to be retransmitted because of subscriber mobility in the separate addressing model.

VARIABLE: dataseg
TYPE: integer
ALIASES: none
USED: removefirstevent, start, loop, insertevent
FUNCTION: number of data segments outstanding

VARIABLE: datause
TYPE: real
ALIASES: none
USED: printstats
FUNCTION: percent channel is in use transmitting data

VARIABLE: destination
TYPE: integer
ALIASES: none
USED: start, initialize
FUNCTION: indicates subscriber that is the receiver in the connection

VARIABLE: done
TYPE: boolean
ALIASES: none
USED: start, loop, initialize
FUNCTION: indicates that no more data segments should be transmitted

VARIABLE: effectivethroughput
TYPE: real
ALIASES: none
USED: printstats
FUNCTION: number of data bits per second transmitted

VARIABLE: event
TYPE: state
ALIASES: none
USED: removefirstevent, loop
FUNCTION: indicates what state the system is in when the event occurs

VARIABLE: exchange
TYPE: integer
ALIASES: none
USED: start
FUNCTION: used to switch source and destination nodes

VARIABLE: flag
TYPE: boolean
ALIASES: none
USED: start, main, insertevent, loop, removefirstevent
FUNCTION: indicates that this event is new

VARIABLE: first
TYPE: elemptr
ALIASES: none
USED: nextnode, insertevent, removefirstevent, start, initialize, loop
FUNCTION: pointer to element record

VARIABLE: holding
TYPE: real
ALIASES: none
USED: subservice
FUNCTION: used to accumulate subservice time during calculation

VARIABLE: host
TYPE: integer
ALIASES: none
USED: nextnode, removefirstevent, loop, start, insertevent
FUNCTION: subscriber the message is headed for

VARIABLE: i
TYPE: integer
ALIASES: none
USED: adddestination, printstats, loop
FUNCTION: represents a queue number

VARIABLE: it
TYPE: integer
ALIASES: none
USED: subservice
FUNCTION: used as the counting variable in a for loop

VARIABLE: j
TYPE: integer
ALIASES: none
USED: adddestination, loop
FUNCTION: represents a queue number

VARIABLE: l
TYPE: pointer
ALIASES: none
USED: insertevent
FUNCTION: an elemptr used to step through the event list

VARIABLE: last
TYPE: elemptr
ALIASES: none
USED: insertevent, removefirstevent, initialize
FUNCTION: pointer to element record

VARIABLE: location
TYPE: integer
ALIASES: none
PART OF: subscriber
USED: nextnode, initialize, loop
FUNCTION: indicatgs where a subscriber is located

VARIABLE: meanconnectinterarrival
TYPE: real
ALIASES: none
USED: start, initialize
FUNCTION: average connection length

VARIABLE: meandatasize
TYPE: integer
ALIASES: none
USED: initialize, printstats, delay
FUNCTION: average data segment size

VARIABLE: meaninterarrival
TYPE: integer
ALIASES: pnone
USED: start, initialize
FUNCTION: mean time between connections

VARIABLE: mnsvc
TYPE: real
ALIASES: none
USED: subservice
FUNCTION: used to hold meanservice time passed to
subservice function

VARIABLE: movedflag
TYPE: boolean
ALIASES: none
PART OF: subscriber
USED: nextnode, initialize, loop
FUNCTION: indicates that a subscriber is movyng

VARIABLE: movetime
TYPE: real
ALIASES: none
USED: start, loop
FUNCTION: time at which the subscriber will move

VARIABLE: n
TYPE: integer
ALIASES: none
USED: subservice
FUNCTION: number of servers for a particular node

VARIABLE: newlocation
TYPE: integer
ALIASES: none
USED: nextnode, initialize, loop
FUNCTION: indicates where a subscriber is moving

```
VARIABLE:  nodes
TYPE:  array of records
ALIARES* fone
COMPOSED OF:  routing
USED:  nextnode, adddestination
FUNCTION:  contains information to route message through
           queues


VARIABLE:  numberevents
TYPE:  integer
ALIASES:  none
USED:  printstats, initialize, loop
FUNCTION:  number of events completed


VARIABLE:  overallthroughput
TYPE:  real
ALIASES:  none
USED:  printstats
FUNCTION:  total number of bits transmitted per second


VARIABLE:  propdelay
TYPE:  real
ALIASES:  none
USED:  initialize, printstats, delay
FUNCTION:  delay in channel


VARIABLE:  q
TYPE:  integer
ALIASES:  none
USED:  nextnode, endcycle, detcurrentevent, insertevent,
       arrive, complete, removefirstevent
FUNCTION:  queue number


VARIABLE:  rate
TYPE:  real
ALIASES:  none
USED:  initialize, printstats, delay
FUNCTION:  speed of transmission data


VARIABLE:  retransmit
TYPE:  integer
ALIASES:  none
USED:  nextnode, printstats, initialize
FUNCTION:  number of times a connection was broken because
           of a mobile subscriber in the separate
           addressing scheme
```

VARIABLE: routing
TYPE: routingptr
ALIASES: none
USED: nextnode, adddestination
FUNCTION: points to next queue in route

VARIABLE: service
TYPE: real
ALIASES: none
USED: subservice
FUNCTION: service time calculated for each server

VARIABLE: servicetime
TYPE: real
ALIASES: none
USED: arrive
FUNCTION: time an event will spend in a queue after arriving at that queue

VARIABLE: source
TYPE: integer
ALIASES:none
USED: start, initialize, nextnode
FUNCTION: indicates subscriber that originated the connection

VARIABLE: subscriber
TYPE: array of record
ALIASES: none
COMPOSED OF: location, newlocation, movedflag
USED: nextnode, initialize, loop
FUNCTION: used to indicate which node a subscriber is currently attached to, where its next location is, and if it's currently moving

VARIABLE: temp
TYPE: ptr
ALIASES: none
USED: insertevent, removefirstevent, adddestination
FUNCTION: an elemptr used in inserting an event into the event list

VARIABLE: tempclock
TYPE: real
ALIASES: none
USED: start, loop
FUNCTION: time the connection should be stopped

VARIABLE: time
TYPE: real
ALIASES: none
USED: removefirstevent, insertevent
FUNCTION: time that event should occur

VARIABLE: totalbits
TYPE: integer
ALIASES: none
USED: delay
FUNCTION: number of bits in the message, both data and header

VARIABLE: totaldataseg
TYPE: integer
ALIASES: none
USED: insertevent, printstats, initialize
FUNCTION: total number of data segments transmitted during simulation

VARIABLE: totaldataoctets
TYPE: integer
ALIASES: none
USED: insertevent, printstats, initialize
FUNCTION: total number of segments transmitted during simulation

VARIABLE: totallength
TYPE: integer
ALIASES: none
USED: initialize, loop
FUNCTION: length of the element record list

VARIABLE: totalsegments
TYPE: integer
ALIASES: none
USED: insertevent, printstats, initialize
FUNCTION: total number of segments transmitted during simulation

VARIABLE: wait
TYPE: boolean
ALIASES: none
USED: start, loop
FUNCTION: indicates the system is waiting on a transmitted data segment to be received before the connection can be closed

VARIABLE: x
TYPE: integer
ALIASES: none
USED: min
FUNCTION: used to determine which is the smaller of two numbers passed to the function min

VARIABLE: y
TYPE: integer
ALIASES: none
USED: min
FUNCTION: used to determine which is the smaller of two numbers passed to the function min

VARIABLE: z
TYPE: integer
ALIASES: none
USED: subservice, start, arrive, initialize, loop
FUNCTION: used as a seed in the system function random

Modules

MODULE: adddestination
TYPE: procedure
CALLING MODULES: initialize
MODULES CALLED: none
FUNCTION: creates routing table

MODULE: arrive
TYPE: procedure
CALLING MODULES: loop
MODULES CALLED: insertevent, min, subservice, delay
FUNCTION: simulates an event arriving at a queue

MODULE: complete
TYPE: procedure
CALLING MODULES: loop
MODULES CALLED: min
FUNCTION: simulates an event completing service at a node

MODULE: delay
TYPE: function
CALLING MODULES: start
MODULES CALLED: none
FUNCTION: calculates the delay due to overhead of TCP/IP headers

MODULE: detcurrentevent
TYPE: function
CALLING MODULES: insertevent, loop
MODULES CALLED: none
FUNCTION: determines what the currentevent should indicate given a certain queue number y

MODULE: endcycle
TYPE: function
CALLING MODULES: loop
MODULES CALLED: min
FUNCTION: keeps track of the number of simulation connections

MODULE: getnext1
TYPE: function
CALLING MODULES: nextnode
MODULES CALLED: none
FUNCTION: determines nextnode for subscriber 1

MODULE: getnext2
TYPE: function
CALLING MODULES: nextnode
MODULES CALLED: none
FUNCTION: determines nextnode for subscriber 2

MODULE: initialize
TYPE: procedure
CALLING MODULES: main
MODULES CALLED: adddestination
FUNCTION: initializes variables and routing table

MODULE: insertevent
TYPE: procedure
CALLING MODULES: arrive, start
MODULES CALLED: none
FUNCTION: inserts event into event list according to time for event to occur

MODULE: loop
TYPE: procedure
CALLING MODULES: main
MODULES CALLED: nextnode, arrive, complete, removefirstevent, insertevent, detcurrentevent, subservice, endcycle
FUNCTION: main loop of simulation program

MODULE: main
TYPE: program
CALLING MODULES: system
MODULES CALLED: initialize, start, loop, printstats
FUNCTION: simulate network

MODULE: min
TYPE: function
CALLING MODULES: arrive, complete
MODULES CALLED: none
FUNCTION: returns smaller of two numbers input

MODULE: nextnode
TYPE: function
CALLING MODULES: loop
MODULES CALLED: getnext1, getnext2
FUNCTION: determines the next queue in the routing of events through the system

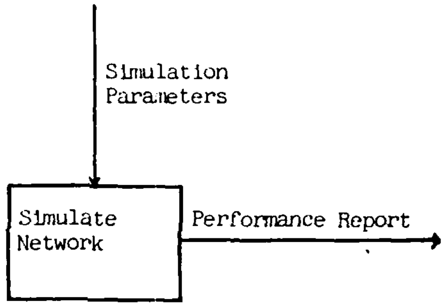
MODULE: printstats
TYPE: procedure
CALLING MODULES: main
MODULES CALLED: none
FUNCTION: calculates and prints out statistics

MODULE: removefirstevent
TYPE: procedure
CALLING MODULES: loop
MODULES CALLED: none
FUNCTION: removes the first event from the event list

MODULE: start
TYPE: procedure
CALLING MODULES: main
MODULES CALLED: insertevent
FUNCTION: startup procedures for a connection

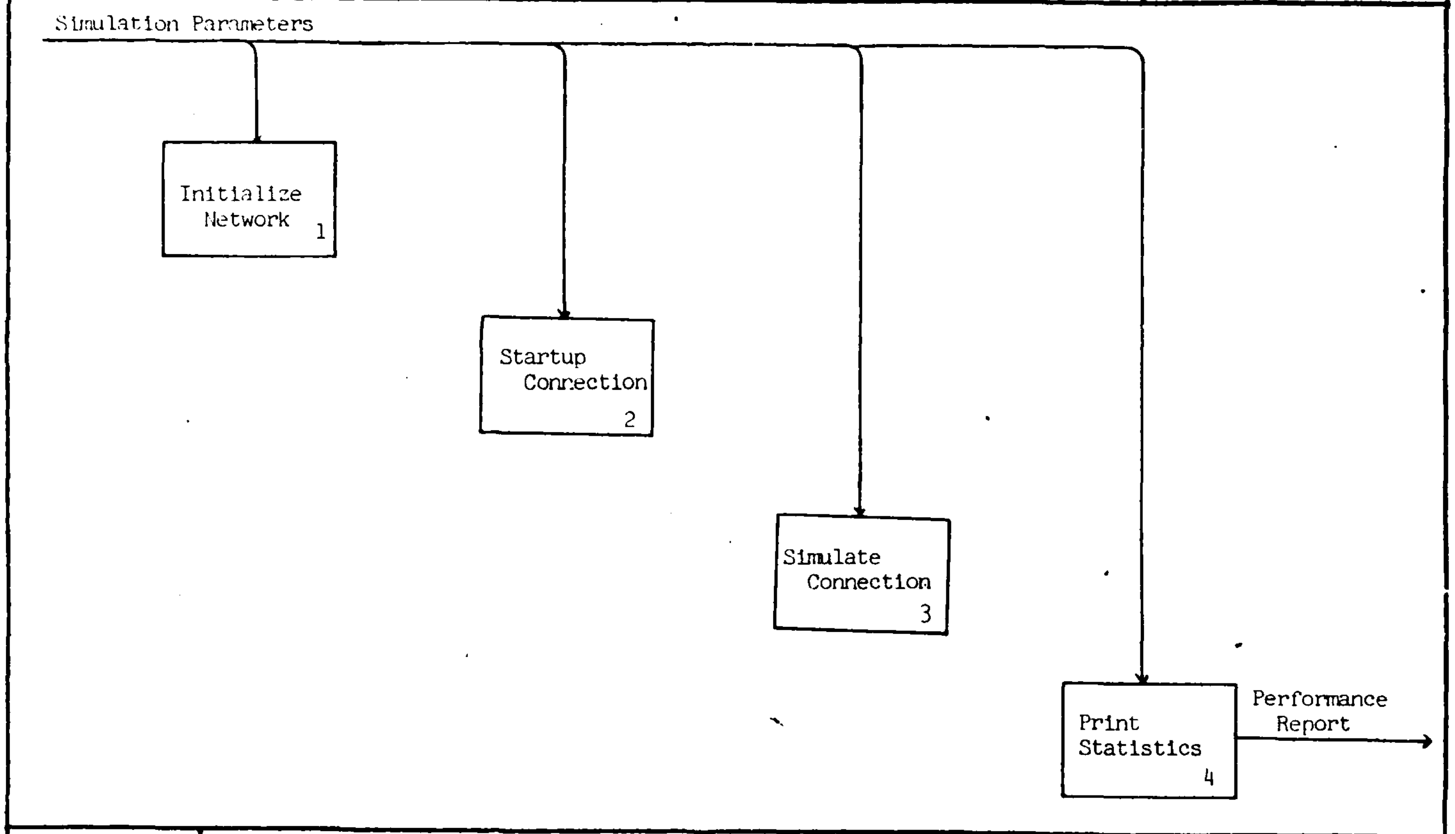
MODULE: subservice
TYPE: function
CALLING MODULES: arrive, loop
MODULES CALLED: none
FUNCTION: calculates n exponentail values for an n-stage
          Erlang distribution

AUTHOR: Capt Bill M. Husk	DATE: 6 Nov 84	READER				
PROJECT: Network Simulation	REV: 1.0	DATE				



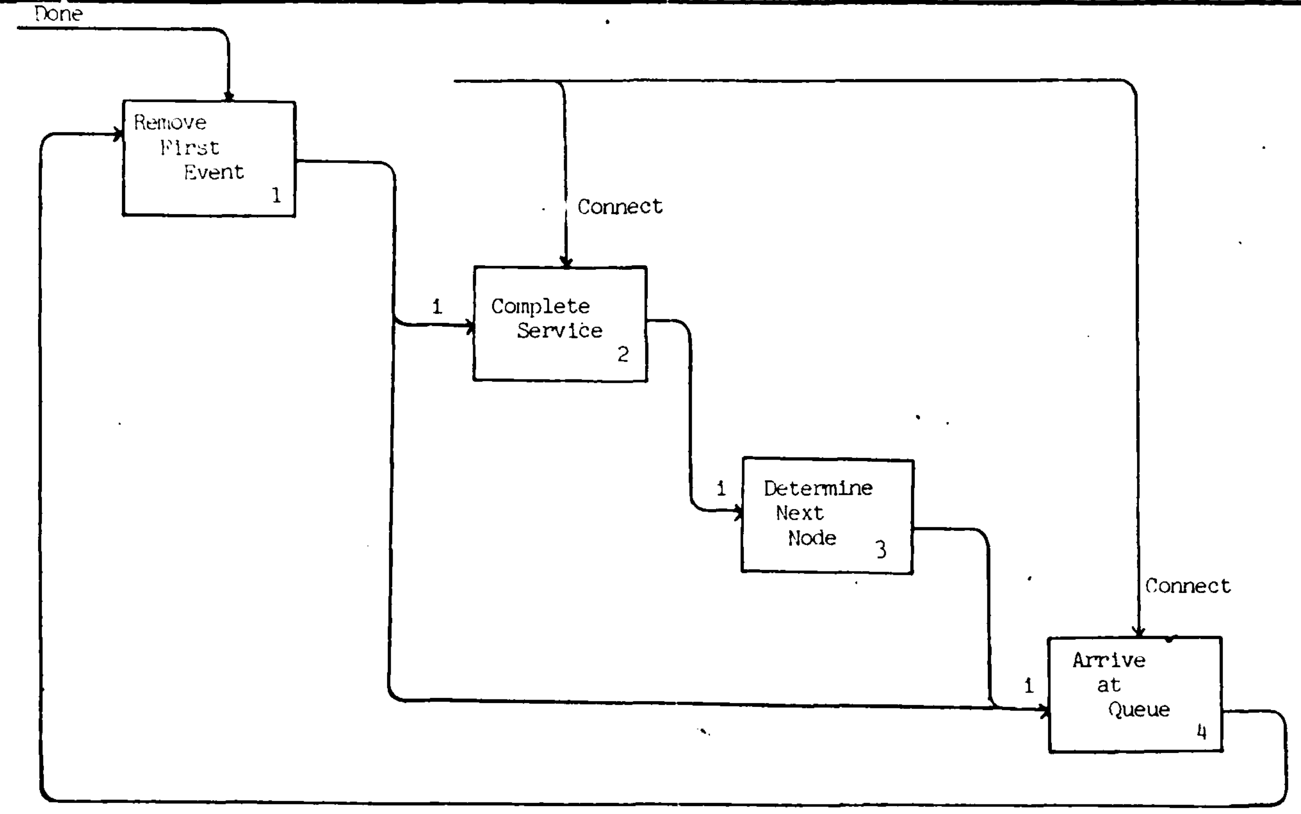
NODE: A-0	TITLE: Simulate Network	NUMBER: 1

AUTHOR: Capt Gill M. Musa	**DATE:** 6 Nov 84	**READER**				
PROJECT: Network Simulation	**REV:** 1.0	**DATE**				



NODE: A0	**TITLE:** Network Simulation	**NUMBER:** 2

AUTHOR: Capt Gail M. Nusz	**DATE:** 6 Nov 84	**READER**				
PROJECT: Network Simulation	**REV:** 1.0	**DATE**				



NODE: A3	**TITLE:** Simulate Connection	**NUMBER:** 3

APPENDIX B: Code

This appendix contains the code for the baseline model. All three simulations programs are based on this code.

```
(*****************************************************************
DATE:06 November 1984
VERSION:  1.0
TITLE:  Simulate
FILENAME:  thesis.p
OWNER:  Capt. Gail M. Nusz
SOFTWARE SYSTEM:  Network Simulation
OPERATING SYSTEM:  VAX/Unix
LANGUAGE:  Pascal
CONTENTS:  min, subservice, nextnode, endcycle, adddestination,
           insertevent, removefirstevent, delay, start, arrive,
           complete, printstats, initialize, detcurrentevent,
           getnext1, getnext2.
USE:       The program "change" must be executed first.  Change
           is an interactive program to load the file "paramfile"
           with variable parameters to be used by the simulation.
           To run the simulation the file "base" should be
           executed.
FUNCTION:  To simulate a mobile packet-switching network with
           TCP/IP as protocols.
*****************************************************************)


program simulate(input,output,paramfile);
  const
        nq=36;
        nio=1; (*number of servers per queue*)
        nn=36; (*number of nodes*)
        aot1=1; aot2=2; adt1=3; act1=4; act2=5; act3=6; act4=7;
        aor1=8; aor2=9; adr1=10; acr1=11; acr2=12; acr3=13; acr4=14;
        bot1=15; bot2=16; bdt1=17; bct1=18; bct2=19; bct3=20; bct4=21;
        bor1=22; bor2=23; bdr1=24; bcr1=25; bcr2=26; bcr3=27; bcr4=28;
        s1=29; r1=30; s2=31; r2=32; s3=33; r3=34; net1=35; net2=36;
  type
        state=(connectreq,connectack,datasnd,closereq,closewait1,
                closewait2,closedelayt,connectacc,connectest,
                datarec,closewait,closeacc,closeack,closedelayr);
        (*state of connection*)
        elemptr= ^element;
        element=record (*used to store event information*)
                time:real; (*time that event should occur*)
                param:integer; (*queue to which event belongs*)
                new:boolean; (*event was generated from outside*)
                segclock:real; (*time to transmit*)
                dest:integer; (*destination host*)
                eventtype:state; (*function of node*)
                dataoctets:integer; (* # of octets in segment*)
                next:elemptr  (*next event in event list*)
                end;
        routingptr= ^routingelement;
        routingelement=record (*used in determining nextnode*)
                destination: 1..nq; (*next queue to arrive at*)
                nextrouting: routingptr  (*next queue in list*)
                end;
```

```
var   z:integer;      (*used as seed for random number generator*)
      i:integer;
      j:integer;
      numruns:integer;
      ns:integer;
      overheadoctets:integer;
      tcps:real;
      tcpr:real;
      ips:real;
      ipr:real;
      net1ser:real;
      net2ser:real;
      paramfile:text;
      flag:boolean; (*indicates an event arrived from outside system*)
      done:boolean; (*indicates that no more data segments send*)
      dataseg:integer; (*number of data segments outstanding*)
      wait:boolean;  (*system waiting to close on data segment*)
      datacumclock:real; (*time spend transmitting data segments*)
      cumclock:real; (*time spend transmitting segments*)
      tempsegclock:real; (*temp clock for cumclock and datacumclock*)
      tempclock:real;  (*time connection should be stopped*)
      dataarrival:real; (*time between data segment accept & data bld*)
      meanconnectinterarrival:real; (*mean connection time*)
      service:real; (*temporary variable to check service time*)
      first,last,avail:elemptr; (*pointers to element record*)
      availrouting: routingptr; (*pointer to routingelement record*)
      clock:real; (*keeps track of total run time*)
      meandatasize:integer; (*average data segment size*)
      rate:real; (*speed of transmitting data over comm. medius*)
      propdelay:real; (*delay in comm. medium*)
      totaldataoctets:integer; (*number of octets in data segment*)
      totalsegments:integer; (*number of segments transmitted*)
      totaldataseg:integer; (*number of data segments transmitted*)
      arrivalflag:boolean; (*indicates to insertevent event from arrive
      currentdataoctets:integer; (*number of octets in current event*)
      cycles:integer; (*number of cycles that have completed*)
      host:integer; (*who the segment is headed for*)
      event:state; (*state of connection*)
      currentevent:state; (*present state of connection*)
      source:integer; (*who the originator of connection is*)
      destination:integer; (*who the receiver of connection is*)
      queues:array[1..nq] of (*characteristics of queue*)
              record
                numberservers:integer;
                meanservice:real;
                removaltime:real;
                length:integer;
                timelengthchanged:real;
                sumbusytime:real;
                numbercompletions:integer;
                bt:real;
                nc:real;
              end;
```

```
nodes: array[1..nn] of  (*queue/server system*)
        record
                routing: routingptr
        end;
subscriber: array[1..3] of
        record
                location: integer
        end;
meaninterarrival:  real; (*mean time between connections*)
run, numberevents: integer;
timecyclestarted:real;
totallength: integer;
```

```
(**************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  min
MODULE NUMBER:
FUNCTION:  returns the smaller of two numbers input.
INPUTS:  x,y
OUTPUTS:  min
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  arrive, complete
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
**************************************************************)



function min(var x,y:integer):integer;
  (*determines the smaller of two numbers and returns smaller*)
  begin
    if x<=y then
      min:=x
    else
      min:=y
  end; (*min*)
```

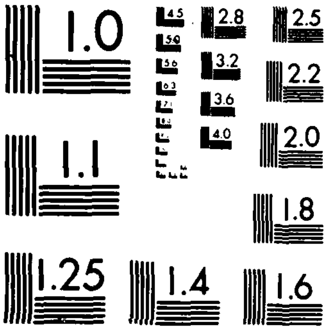
```
(****************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  subservice
MODULE NUMBER:
FUNCTION:  calculates n exponential values for an n-stage
           Erlang distribution.
INPUTS:  mnsvc, n
OUTPUTS:  subservice
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  arrive, main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
****************************************************************)


function subservice(mnsvc:real;n:integer):real;
(*subservice gets n exponential values for an n-stage*)
(*Erlang distribution*)
(*service time is not purely exponential, we do not *)
(*allow service time to be shorter than the mean    *)
var it:integer;
    holding:real;
    service:real;
begin
  holding:=0.0;
  for it:=1 to n do
    begin
          service:=-1.0*mnsvc*ln(random(z));
          if service<mnsvc then
            holding:=holding-mnsvc
          else holding:=holding-service;
    end;
  subservice:=holding;
end; (*subservice*)
```

```
(*************************************************************
DATE:  30 OCT 84
VERSION:  1.0
NAME:  getnext1
MODULE NUMBER:
FUNCTION:  determines nextnode for subscriber 1.
INPUTS:  none
OUTPUTS:  nextnode
GLOBAL VARIABLES USED:  currentevent
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  nextnode
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
**************************************************************)
function getnext1:integer;
begin
        case currentevent of
                connectreq: getnext1:=aor1;
                connectack: getnext1:=aor2;
                datasnd:    getnext1:=adr1;
                closereq:   getnext1:=acr1;
                closewait1: getnext1:=acr2;
                closewait2: getnext1:=acr3;
                closedelayt: getnext1:=acr4
        end
end;(*getnext1*)
```

```
(**************************************************************
DATE:  30 OCT 84
VERSION:  1.0
NAME: getnext2
MODULE NUMBER:
FUNCTION:  determines nextnode for subscriber 2.
INPUTS:  none
OUTPUTS:  nextnode
GLOBAL VARIABLES USED:  currentevent
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  nextnode
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
**************************************************************)
function getnext2:integer;
begin (*getnext2*)
        case currentevent of
                connectreq:  getnext2:=bor1;
                connectack:  getnext2:=bor2;
                datasnd:     getnext2:=bdr1;
                closereq:    getnext2:=bcr1;
                closewait1:  getnext2:=bcr2;
                closewait2:  getnext2:=bcr3;
                closedelayt: getnext2:=bcr4
        end
end; (*getnext2*)
```

AD-A174 268 ADDRESSING FOR MOBILE SUBSCRIBERS IN A PACKET-SWITCHING 2/2
MULTI-NETWORK ENVIRONMENT(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI G M NUSZ
UNCLASSIFIED DEC 84 AFIT/GCS/ENG/84D-19 F/G 17/2 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

```
(**************************************************************
DATE:  30 OCT 84
VERSION:  2.0
NAME:  nextnode
MODULE NUMBER:
FUNCTION:  determines the next queue in the routing of events
           through the system.
INPUTS:  q, host
OUTPUTS:  nextnode
GLOBAL VARIABLES USED:  nodes[j], currentevent
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:   none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
**************************************************************)


function nextnode(q:integer;host:integer):integer;
(*finds the next node for job q to go to*)
(*some routing decisions are based on current *)
(*event state and who the receiving host is   *)
var
    temp:routingptr;
begin
 if ((q=aor1)or(q=aor2)or(q=adr1)or(q=acr1)or(q=acr2)or(q=acr3)
       or(q=bor1)or(q=bor2)or(q=bdr1)or(q=bcr1)or(q=bcr2)or(q=bcr3))
    then
(*not a decision point, easy routing decision*)
  begin
   temp:=nodes[q].routing;
   if temp=nil then
    begin
     writeln('nextnode - undefined routing from node',q);
     halt
    end;
   nextnode:=temp^.destination;
  end
 else if ((q=acr4)or(q=bcr4)) then
       (*at end of connection*)
      nextnode:=99
     else if q=net1 then
           begin
            if subscriber[host].location=1 then
               nextnode:=r1
            else nextnode:=r2;
           end
```

```
else if q=net2 then
      begin
       if subscriber[host].location=3 then
             nextnode:=r3
       else nextnode:=r2;
      end
     else if q=s2 then
           begin
             if subscriber[host].location=1 then
              nextnode:=net1
             else if subscriber[host].location=2 then
                     nextnode:=r2
                  else nextnode:=net2;
           end
          else if ((q=r1)or(q=r3)) then
                begin
                 case host of
                     1: nextnode:=getnext1;
                     2: nextnode:=getnext2
                 end
                end
               else if q=r2 then
                     begin
                      if subscriber[host].location<>2 then
                        nextnode:=s2
                      else
                       case host of
                             1:nextnode:=getnext1;
                             2:nextnode:=getnext2
                       end
                     end
                    else if q=s1 then
                          begin
                           if subscriber[host].location=1 then
                             nextnode:=r1
                           else nextnode:=net1;
                          end
                         else if q=s3 then
                               begin
                                if subscriber[host].location=3 then
                                  nextnode:=r3
                                else nextnode:=net2;
                               end
                     else if ((q=aot1)or(q=aot2)or(q=adt1)
                             or(q=act1)or(q=act2)or(q=act3)
                             or(q=act4)) then
                           case subscriber[1].location of
                             1: nextnode:=s1;
                             2: nextnode:=s2;
                             3: nextnode:=s3
                           end
```

```
                                   else
                                    case subscriber[2].location of
                                      1: nextnode:=s1;
                                      2: nextnode:=s2;
                                      3: nextnode:=s3
                                    end;
end; (*nextnode*)
```

```
(**************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  endcycle
MODULE NUMBER:
FUNCTION:  keeps track of number of simulated connections.
INPUTS:  none
OUTPUTS:  endcycle
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  queues[q]
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  min
CALLING MODULES:  main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
**************************************************************)


function endcycle: boolean;
(*determines whether at end of connection cycle
  if so, endcycle updates accumulators*)
  var q: integer;
  begin
    if cycles>numruns then
      begin
        endcycle:=true;
        timecyclestarted:=clock;
        for q:=1 to nq do
          with queues[q] do
            begin
              sumbusytime:=(sumbusytime
                             +(clock-timelengthchanged)
                             *min(length,numberservers))/numberservers;
              timelengthchanged:=clock;
              bt:=bt+sumbusytime;
              nc:=nc+numbercompletions;
              sumbusytime:=0.0;
              numbercompletions:=0;
            end
      end
    else
      endcycle:=false
  end; (*endcycle*)
```

```
(****************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  adddestination
MODULE NUMBER:
FUNCTION:  creates routing table.
INPUTS:  i, j
OUTPUTS:  none
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  nodes[i], availrouting
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  initialize
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
****************************************************************)



procedure adddestination(i,j:integer);
(*adds destination node j to routing list for node i*)
var temp: routingptr;
  begin
    if availrouting=nil then
      new(temp)
    else
      begin
        temp:=availrouting;
        availrouting:=availrouting^.nextrouting
      end;
    temp^.destination:=j;
    temp^.nextrouting:=nodes[i].routing;
    nodes[i].routing:=temp
  end; (*adddestination*)
```

```
(****************************************************************
DATE:  27 AUG 84
VERSION:  1.0
NAME:  detcurrentevent
MODULE NUMBER:
FUNCTION:  determines what the currentevent should indicate
           given a certain queue number y.
INPUTS:  q
OUTPUTS:  none
GLOBAL VARIABLES USED: currentevent
GLOBAL VARIABLES CHANGED: none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  insertevent, main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
****************************************************************)

function detcurrentevent(q:integer):state;
begin (*detcurrentevent*)
                case q of
                        aot1,bot1: detcurrentevent:=connectreq;
                        aor1,bor1: detcurrentevent:=connectacc;
                        aot2,bot2: detcurrentevent:=connectack;
                        aor2,bor2: detcurrentevent:=connectest;
                        adt1,bdt1: detcurrentevent:=datasnd;
                        adr1,bdr1: detcurrentevent:=datarec;
                        act1,bct1: detcurrentevent:=closereq;
                        acr1,bcr1: detcurrentevent:=closewait;
                        act2,bct2: detcurrentevent:=closewait1;
                        acr2,bcr2: detcurrentevent:=closeacc;
                        act3,bct3: detcurrentevent:=closewait2;
                        acr3,bcr3: detcurrentevent:=closeack;
                        act4,bct4: detcurrentevent:=closedelayt;
                        acr4,bcr4: detcurrentevent:=closedelayr;
                  s1,r1,s2,r2,s3,r3,net1,net2: detcurrentevent:=currentevent
                end;
end; (*detcurrentevent*)
```

```
(**************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  insertevent
MODULE NUMBER:
FUNCTION:  inserts event into event list according to time,
           time for event to occur.
INPUTS:  time, q, flag
OUTPUTS:  none
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  totalsegments, totaldataseg
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  arrive, main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
**************************************************************)


procedure insertevent(time:real;q:integer;flag:boolean);
(*insertevent adds event at time  for param q to list*)
(*if flag is true, new is updated to true*)
  var temp,l: elemptr;
  begin
     if avail=nil then
      new(temp)
     else
      begin (*previously used storage available*)
        temp:=avail;
        avail:=avail^.next
      end;
     temp^.time:=time;
     (*updates the segment clock when at start of new pass through*)
     (*network                                                    *)
     if ((q=aot1)or(q=aot2)or(q=adt1)or(q=act1)or(q=act3)or(q=act4)
        or(q=bot1)or(q=bot2)or(q=bdt1)or(q=bct1)or(q=bct2)or(q=bct3)
        or(q=bct4)) then
        temp^.segclock:=clock
     else temp^.segclock:=tempsegclock;
     temp^.param:=q;
     temp^.new:=flag;
     temp^.dataoctets:=currentdataoctets;
     temp^.dest:=host;
     (*determine eventtype*)
     temp^.eventtype:=detcurrentevent(q);
  (*add event to list*)
   if first = nil then
     begin (*list is empty*)
      first:=temp;
      last:=temp;
      temp^.next:=nil
     end
   else if time<first^.time then
      begin (*insert at beginning*)
        temp^.next:=first;
        first:=temp
      end
   else if time>=last^.time then
      begin (*insert at end of list*)
        last^.next:=temp;
        last:=temp;
        temp^.next:=nil
      end
   else
      begin (*insert somewhere in middle*)
        l:=first;
        while time>=l^.next^.time do
          l:=l^.next;
        temp^.next:=l^.next;
        l^.next:=temp
      end;
  (*update statistics*)
  if arrivalflag then
      begin
        if ((q=aot1)or(q=aot2)or(q=adt1)or(q=act1)or
              (q=act2)or(q=act3)or(q=act4)or(q=bot1)or(q=bot2)
              or(q=bdt1)or(q=bct1)or(q=bct2)or(q=bct3)or
              (q=bct4)) then
              totalsegments:=totalsegments+1;
        if ((q=adt1)or(q=bdt1)) then
          begin
              dataseg:=dataseg+1;
              totaldataoctets:=totaldataoctets+currentdataoctets;
          end
        else if ((q=adr1)or(q=bdr1)) then
              totaldataseg:=totaldataseg+1;
      end;
end; (*insertevent*)
```

```
(**************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  removefirstevent
MODULE NUMBER:
FUNCTION:  removes the first event from the event list
INPUTS:  time
OUTPUTS:  event, q, host
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  first, datacumclock
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
**************************************************************)

procedure removefirstevent(var time:real; var q:integer;
                           var event:state; var host:integer);
(*removefirstevent returns time t param q e state and h host of first
  event*)
  var temp: elemptr;
  begin
    if first=nil then
      begin
        writeln('removefirstevent -- empty list');
        halt
      end
    else
      begin (*get information from event record*)
        if ((q=adr1)or(q=bdr1)) then
          dataseg:=dataseg-1;
        time:=first^.time;
        q:=first^.param;
        flag:=first^.new;
        host:=first^.dest;
        event:=first^.eventtype;
        currentdataoctets:=first^.dataoctets;
        tempsegclock:=first^.segclock;
        (*update statistics*)
        if ((q=aor1)or(q=aor2)or(q=adr1)or(q=acr1)or(q=acr2)
                or(q=acr3)or(q=acr4)or(q=bor1)or(q=bor2)or(q=bdr1)or
                (q=bcr1)or(q=bcr2)or(q=bcr3)or(q=bcr4)) then
          begin
                cumclock:=cumclock+clock-first^.segclock;
                if ((q=adr1)or(q=bdr1)) then
                  datacumclock:=datacumclock+clock-first^.segclock;
          end;
        (*put element storage on available list*)
        temp:=first;
        first:=first^.next;
        if first=nil then
          last:=nil;
        temp^.next:=avail;
        avail:=temp
      end
end; (*removefirstevent*)
```

```
(*****************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  delay
MODULE NUMBER:
FUNCTION:  calculates delay from overhead in TCP/IP headers.
INPUTS:  currentdataoctets
OUTPUTS:  delay
GLOBAL VARIABLES USED:  currentdataoctets
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  arrive
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
*****************************************************************)


function delay(currentdataoctets:integer):real;
(*calculates delay through communication medium*)
var totalbits:integer;
begin (*calculated delay through channels*)
  totalbits:=8*(currentdataoctets+overheadoctets);
  delay:=(1.0/rate*totalbits)+propdelay;
end; (*delay*)
```

```
(*****************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  start
MODULE NUMBER:
FUNCTION:  startup procedures for a connection
INPUTS:  none
OUTPUTS:  none
GLOBAL VARIABLES USED:  flag
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  insertevent
CALLING MODULES:  main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
*****************************************************************)
procedure start;
(*initialization for each connection*)
var exchange,x:integer;
begin
  dataseg:=0;
  wait:=false;
  if first<>nil then
        writeln(' event list not empty',first^.param,first^.time);
  cycles:=cycles+1; (*increment number of connections*)
  flag:=true;  (*set flag to indicate event from outside*)
  done:=false; (*set connection to allow data transfers*)
  exchange:=source; (*switch source and destination hosts*)
  source:=destination;
  destination:=exchange;
  host:=destination;
  (*determine which node to start connection at*)
  if source=1 then
        x:= aot1
  else
        x:=bot1;
  tempclock:=clock-subservice(meanconnectinterarrival,1);
  insertevent(tempclock,x,flag);
  (*determine which node to finish connection at*)
  if source=1 then
        x:=act1
  else
        x:=bct1;
  tempclock:=tempclock-meaninterarrival*ln(random(z));
  insertevent(tempclock,x,flag);
  currentevent:=connectreq; (*set current event state to beginning*)
  currentdataoctets:=0;
end; (*start*)
```

```
(****************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  arrive
MODULE NUMBER:
FUNCTION:  simulates an event starting.
INPUTS:  q
OUTPUTS:  none
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  queues[q], arrivalflag
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  insertevent, subservice, delay
CALLING MODULES:  main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
****************************************************************)


procedure arrive(q:integer);
(*handles arrival of a job at queue q*)
var servicetime:real;
  begin
    with queues[q] do
      begin
        (*stastics*)
        sumbusytime:=sumbusytime+(clock-timelengthchanged)
                       *min(length,numberservers);
        timelengthchanged:=clock;
        (*mechanics*)
        length:=length+1;
        arrivalflag:=true;
        if length<=numberservers then
          begin
            if ((q=net1)or(q=net2)) then
                begin
                 removaltime:=clock-subservice(meanservice,numberservers)
                                +delay(currentdataoctets);
                 insertevent(removaltime,q,false)
                end
             else
                 begin
                      servicetime:=-1.0*meanservice*ln(random(z));
                      if servicetime<meanservice then
                       begin
                        removaltime:=clock+meanservice;
                        insertevent(removaltime,q,false)
                       end
```

```
                    else
                      begin
                        removaltime::=clock+servicetime;
                        insertevent(removaltime,q,false);
                      end
                 end;
        end
      else
              begin
                if ((q=net1)or(q=net2)) then
                      insertevent(removaltime-
                        subservice(meanservice,numberservers)
                          +delay(currentdataoctets),q,false)
                else
                  begin
                   servicetime:=-1.0*meanservice*ln(random(z));
                   if servicetime<meanservice then
                     insertevent(removaltime+meanservice,q,false)
                   else insertevent(removaltime+servicetime,q,false);
                  end;
              end;
      arrivalflag:=false;
    end
end; (*arrive*)
```

```
(**************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  complete
MODULE NUMBER:
FUNCTION:  simulates an event being completed.
INPUTS:  q
OUTPUTS:  none
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  queues[q]
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
**************************************************************)


procedure complete(q:integer);
(*handles completion of a job at queue q*)
  begin
    with queues[q] do
      begin
        (*stastics*)
        numbercompletions:=numbercompletions+1;
        sumbusytime:=sumbusytime+(clock-timelengthchanged)
                        *min(length,numberservers);
        timelengthchanged:=clock;
        (*mechanics*)
        length:=length-1;
      end;
  end; (*complete*)
```

```
(***************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  printstats
MODULE NUMBER:
FUNCTION:  calculates and prints out statistics.
INPUTS:  none
OUTPUTS:  none
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  queues[q]
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  none
CALLING MODULES:  main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
***************************************************************)


procedure printstats;
var
        effectivethroughput:real;
        overallthroughput:real;
        datause:real;
        avesegmentdelay:real;
        avedatasegmentdelay:real;
begin
      (*print statistics*)
  avesegmentdelay:=0.0;
  effectivethroughput:=0.0;
  avedatasegmentdelay:=0.0;
        writeln;
        writeln('number of events:',numberevents:8,
                ' simulated time:',clock:10:3);
        writeln;
        writeln(
   'queue utilization throughput');
      (*produce point estimates only*)
      for i:=1 to nq do
        with queues[i] do
          if numbercompletions+trunc(nc)>0 then
            begin
                sumbusytime:=sumbusytime+bt*numberservers;
                numbercompletions:=numbercompletions
                                   +trunc(nc);
                sumbusytime:=sumbusytime+
                        min(length,numberservers)*
                        (clock-timelengthchanged);
                writeln(i:5,
                        sumbusytime/(numberservers*clock):12:6,
                        numbercompletions/clock:11:4)
            end;
        effectivethroughput:=(totaldataoctets*8)/cumclock;
        avesegmentdelay:=cumclock/totalsegments;
        overallthroughput:=(8.0*totaldataoctets+overheadoctets
                           *8.0*totaldataseg)/cumclock;
        datause:=effectivethroughput/overallthroughput;
        if totaldataseg<>0 then
        avedatasegmentdelay:=datacumclock/totaldataseg;
        writeln;
        writeln(' number of connections completed = ',cycles-1);
        writeln(' cumclock = ',cumclock:10:3);
        writeln(' mean data size = ',meandatasize);
        writeln(' rate = ',rate:12:3);
        writeln(' propogation delay = ',propdelay:10:7);
        writeln(' effective throughput = ',effectivethroughput:12:3);
        writeln(' overall throughput = ',overallthroughput:12:3);
        writeln(' data use = ',datause:5:2);
        writeln(' average segment delay = ',avesegmentdelay:10:4);
        writeln(' average data segment delay = ',avedatasegmentdelay:10:4);
        writeln(' total data octets = ',totaldataoctets);
        writeln(' total segments = ',totalsegments);
        writeln(' total data segments = ',totaldataseg);
end; (*printstats*)
```

```
(*****************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  initialize
MODULE NUMBER:
FUNCTION:  initializes variables and routing table.
INPUTS:  none
OUTPUTS:  none
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  adddestination
CALLING MODULES:  main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
*****************************************************************)


procedure initialize;
  (*initialization*)
var i:integer;
    y:real;
    sendnode1:integer;
    sendnode2:integer;
begin
  z:=3141;     (*an arbitrary value*)
  avail:=nil;  (*no records available*)
  availrouting:=nil; (*no records available*)
  reset(paramfile);
  read(paramfile,numruns);
  read(paramfile,ns);
  read(paramfile,meandatasize);
  read(paramfile,dataarrival);
  read(paramfile,meaninterarrival);
  read(paramfile,meanconnectinterarrival);
  for i:=1 to ns do
    read(paramfile,subscriber[i].location);
  read(paramfile,y);
  read(paramfile,rate);
  read(paramfile,propdelay);
  read(paramfile,overheadoctets);
  read(paramfile,tcps);
  read(paramfile,tcpr);
  read(paramfile,ips);
  read(paramfile,ipr);
  read(paramfile,net1ser);
  read(paramfile,net2ser);
  writeln('datagram size',meandatasize:4,' octets');
  writeln('time between datagrams',dataarrival:4:1,'sec');
writeln('connection length',meaninterarrival:5:1,'sec');
writeln('time between connections',meanconnectinterarrival:5:1);
writeln('transmission rate',rate:10:2,' bps');
writeln('propogation delay',propdelay:8:6,'sec');
writeln('number of octets in headers',overheadoctets:5);
writeln('tcp send service time',tcps:8:6);
writeln('tcp receive service time',tcpr:8:6);
writeln('ip send service time',ips:8:6);
writeln('ip receive service time',ipr:8:6);
writeln('net1 transit time',net1ser:8:6);
writeln('net2 transit time',net2ser:8:6);
writeln('subscriber 1 attached to node',subscriber[1].location:3);
writeln('subscriber 2 attached to node',subscriber[2].location:3);
queues[aot1].meanservice:=tcps; (*service time for servers*)
queues[aor1].meanservice:=tcpr;
queues[aot2].meanservice:=tcps;
queues[aor2].meanservice:=tcpr;
queues[adt1].meanservice:=tcps;
queues[adr1].meanservice:=tcpr;
queues[act1].meanservice:=tcps;
queues[acr1].meanservice:=tcpr;
queues[act2].meanservice:=tcps;
queues[acr2].meanservice:=tcpr;
queues[act3].meanservice:=tcps;
queues[acr3].meanservice:=tcpr;
queues[act4].meanservice:=tcps;
queues[acr4].meanservice:=tcpr;
queues[bot1].meanservice:=tcps:
queues[bor1].meanservice:=tcpr;
queues[bot2].meanservice:=tcps;
queues[bor2].meanservice:=tcpr;
queues[bdt1].meanservice:=tcps;
queues[bdr1].meanservice:=tcpr;
queues[bct1].meanservice:=tcps;
queues[bcr1].meanservice:=tcpr;
queues[bct2].meanservice:=tcps;
queues[bcr2].meanservice:=tcpr;
queues[bct3].meanservice:=tcps;
queues[bcr3].meanservice:=tcpr;
queues[bct4].meanservice:=tcps;
queues[bcr4].meanservice:=tcpr;
queues[s1].meanservice:=ips;
queues[r1].meanservice:=ipr;
queues[s2].meanservice:=ips;
queues[r2].meanservice:=ipr;
queues[s3].meanservice:=ips;
queues[r3].meanservice:=ipr;
queues[net1].meanservice:=net1ser;
queues[net2].meanservice:=net2ser;
for i:=1 to 35 do        (*sets number of servers per queue*)
  queues[i].numberservers:=nio;
queues[net1].numberservers:=4;
queues[net2].numberservers:=4;
```

```
  currentdataoctets:=0;
  totaldataoctets:=0;
  totalsegments:=0;
  totaldataseg:=0;
  cycles:=0;
  done:=false;
  source:=1;
  destination:=2;
  first:=nil;
  last:=nil;
  clock:=0.0;
  cumclock:=0.0;
  datacumclock:=0.0;
  numberevents:=0;
  timecyclestarted:=0.0;
        for i:=1 to nq do
          with queues[i] do  (*initialize queues*)
            begin
             length:=0;
             timelengthchanged:=0.0;
             removaltime:=0.0;
             sumbusytime:=0.0;
             bt:=0.0;
             numbercompletions:=0;
             nc:=0.0;
            end;
if subscriber[1].location=1 then
  sendnode1:=s1
else if subscriber[1].location=2 then
       sendnode1:=s2
     else sendnode1:=s3;
if subscriber[2].location=1 then
  sendnode2:=s1
else if subscriber[2].location=2 then
       sendnode2:=s2
     else sendnode2:=s3;
        adddestination(aot1,sendnode1); (*set up routing between nodes*)
        adddestination(aot2,sendnode1);
        adddestination(adt1,sendnode1);
        adddestination(act1,sendnode1);
        adddestination(act2,sendnode1);
        adddestination(act3,sendnode1);
        adddestination(act4,sendnode1);
        adddestination(bot1,sendnode2);
        adddestination(bot2,sendnode2);
        adddestination(bdt1,sendnode2);
        adddestination(bct1,sendnode2);
        adddestination(bct2,sendnode2);
        adddestination(bct3,sendnode2);
        adddestination(bct4,sendnode2);
        adddestination(aor1,aot2);
        adddestination(aor2,adt1);
        adddestination(adr1,adt1);
```

```
          adddestination(acr1,act2);
          adddestination(acr2,bct3);
          adddestination(acr3,act4);
          adddestination(bor1,bot2);
          adddestination(bor2,bdt1);
          adddestination(bdr1,bdt1);
          adddestination(bcr1,bct2);
          adddestination(bcr2,act3);
          adddestination(bcr3,bct4);
          totallength:=0;
     end; (* initialize *)
```

```
(***************************************************************
DATE:  27 AUG 84
VERSION:  1.0
NAME:  loop
MODULE NUMBER:
FUNCTION:  main loop of simulation program
INPUTS:  none
OUTPUTS:  none
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:
CALLING MODULES:  main
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
***************************************************************)


procedure loop;
var    closeq:integer; (*queue waiting on data segment*)
       dataclock:real; (*temp storage for dataarrival*)
begin (*loop*)
        while (first<>nil) and (not endcycle) do   (*loop for each run*)
          begin
            numberevents:=numberevents+1;
            removefirstevent(clock,i,event,host);  (*get first event*)
            currentevent:=event;
            if flag then (*if begin or end of connection*)
              begin
                arrive(i);
                currentevent:=detcurrentevent(i);
                flag:=false;
                totallength:=totallength+1;
              end
            else (*not beginning or ending of connection*)
              begin
                complete(i);
                if ((i=aor1)or(i=aor2)or(i=adr1)or(i=acr1)or(i=acr2)
                  or(i=acr3)or(i=bor1)or(i=bor2)or(i=bdr1)or(i=bcr1)
                  or(i=bcr2)or(i=bcr3)) then
                  if host=1 then
                    host:=2
                  else host:=1;
                if ((i=act1)or(i=bct1)) then (*if end of data transfer*)
                        done:=true;
                    j:=nextnode(i,host);
                    i:=j;
                    if ((i=adt1)or(i=bdt1)) then
                      begin
                        currentdataoctets:=trunc(-subservice(meandatasize,1));
                      end;
                    if ((i<>99)and(((i<>adt1)and(i<>bdt1))))
                        then
                           begin
                             if ((done)and((i=adt1)or(i=bdt1))) then
                               begin
                                if ((wait)and(dataseg=0)) then
                                 begin
                                  arrive(closeq);
                                  currentevent:=closereq;
                                 end;
                               end
                             else if (((i<>acr2)and(i<>bcr2))or(dataseg=0)) the
                                   begin
                                    arrive(i);
                                    currentevent:=detcurrentevent(i);
                                   end
                                  else
                                   begin
                                    wait:=true;
                                    closeq:=i;
                                   end

                           end
                        else if (((i=adt1)or(i=bdt1))and
                                    ( not done)) then
                                begin
                                 dataclock:=clock-dataarrival*ln(random(z));
                                 if dataclock<tempclock then
                                   insertevent(dataclock,i,true);
                                end
                             else  if i=99 then
                                begin
                                  start;
                                  totallength:=totallength-1;
                                end;
             end
  end;
end; (*loop*)
```

```
(***************************************************************
DATE:  24 AUG 84
VERSION:  1.0
NAME:  main
MODULE NUMBER:
FUNCTION:  simulate network
INPUTS:  none
OUTPUTS:  statistics
GLOBAL VARIABLES USED:  none
GLOBAL VARIABLES CHANGED:  none
GLOBAL TABLES USED:  none
GLOBAL TABLES CHANGED:  none
FILES READ:  none
FILES WRITTEN:  none
MODULES CALLED:  all others
CALLING MODULES:  none
AUTHOR:  CAPT GAIL M. NUSZ
HISTORY:
***************************************************************)


begin (*main*)
        initialize;
        start; (*set up connection*)
        loop;
        printstats;
end.
```

APPENDIX C: Logical Addressing Code Changes

The following lines of code are the changes made to the baseline code to obtain the program for Logical Addressing model.

new variables:

```
movetime:  real;
avemovetime:  real;
numbermoves:  integer;
```

added to procedure start:

```
movetime:=tempclock-avemovetime*ln(random(z));
```

Added to procedure initialize:

```
numbermoves:=0;
```

Added to procedure loop between the lines marked by an "*":

```
*currentevent:=event;
 if clock>movetime then
   begin
     subscriber[2].location:=subscriber[2].location+1;
     movetime:=clock-avemovetime*ln(random(z));
     if subscriber[2].location>3 then
       subscriber[2].location:=1;
     numbermoves:=numbermoves+1;
   end;
*if flag then
```

APPENDIX D: Separate Addressing Code Changes

The following lines of code are changes made to the baseline model to obtain the program for the separate addressing model.

Added variables:

```
datarettransmit: integer;
```

Additions to procedure start:

```
movetime:=tempclock-avemovetime*ln(random(z));
```

Additions to procedure initialize:

```
dataretransmit:=0;
retransmit:=0;
numbermoves:=0;
subscriber[1].movedflag:=false;
subscriber[2].movedflag:=false;
subscriber[1].newlocation:=subscriber[1].location;
subscriber[2].newlocation:=subscriber[2].location;
```

Additions to procedure loop:

```
if clock>movetime then
 begin
   movetime:=clock-avemovetime*ln(random(z));
   subscriber[2].newlocation:=subscriber[2].newlocation+1;
   if subscriber[2].newlocation>3 then
      subscriber[2].newlocation:=1;
   subscriber[2].movedflag:=true;
   numbermoves:=numbermoves+1;
 end;
```

Additions to procedure nextnode:

```
if subscriber[host].movedflag then
 begin
  subscriber[host].location:=subscriber[host].newlocation;
  subscriber[host].movedflag:=false;
  if ((currentevent=datasnd)or(currentevent=datarec)) then
   begin
      dataretransmit:=dataretransmit+1;
      datacumclock:=datacumclock+clock-tempsegclock;
      cumclock:=cumclock+clock-tempsegclock;
      dataseg:=dataseg-1;
      totaldataoctets:=totaldataoctets-currentdataoctets;
      totalsegments:=totalsegments-1;
      if host=1 then
       nextnode:=bdt1
      else nextnode:=adt1;
   end
  else
   begin
      retransmit:=retransmit+1;
      cumclock:=cumclock+clock-tempsegclock;
      totalsegments:=totalsegments-1;
      first:=nil;
      if source=1 then
       begin
        host:=2;
        nextnode:=act1;
       end
      else
       begin
         nextnode:=bct1;
         host:=1;
       end;
   end;
end
```

BIBLIOGRAPHY

1. Kahn, Robert E. and others. "Advances in Packet Radio Technology," Proceedings of the IEEE,66 (11): 1468-1496 (November 1978).

2. Defense Advancdd Research Projects Agency. DoD Standard Transmission Control Protocol. Prepared by Information Sciences Institute, USC, Marina Del Rey, California (September 1981).

3. Varshney, Pramon K. and others. "Simulation Study of a Distributed Packet Radio Network," IEEE Proceedings of Computer Networking Symposium: 99-104 (1981).

4. Sunshine, Carl A. "Addressing Problems in Multi-Network Systems," IEEE Proceedings INFOCOM 82: 12-18 (1982).

5. Sunshine, C. and J. Postel. "Addressing Mobile Hosts in the ARPA Internet Envibonment," IEN-135, USC-ISI (March 1980).

6. Gitman, I. and others. "Routing in Packet-Switching Broadcast Radio Networks," IEEE Transactions on Communications: 926-930 (August 1976).

7. Riordan, J. S. and others. "Access Strategies in Packet Mobile Radio Data Networks," Computer Networks,7: 211-221 (1983).

8. Davies, B. H. and A. S. Bates. "Internetworking in the Military Environment," IEEE ProceedingsIINFOCOM 82: 19-29 (1982).

9. Cerf, Vinton G. and Edward Cain. "the DoD Internet Architecture Model," Computer Networks, 7: 307-318 (1983).

10. Postel, Jonathan B. "Internetwork Protocol Approaches," IEEE Transactions on Communications, COM-28 (4): 604-611 (April 1980).

11. Tannenbaum, Andrew S. Computer Networks. Englewood Cliffs, NJ: Prentice-Hall, Inc. (1981).

12. Defense Advanced Peqearch Projects Agency. DoD Standard Internet Protocol. Prepared by Information Sciences Institute, USC, Marina del Rey, California (September 1981).

13. McQuillan, John M. "Message Addressing Modes for Computer Networks," IEEE Compcon Fall 1978: 80-88 (1978).

14. Shoch, John E. "Inter-Network Naming, Addressing, and Routing," IEEE COMPCON Fall 1978: 72-79 (1978).

15. Boggs, David R. and others. "Pup: An internetwork Architecture," IEEE Transactions on Communications, COM-28 (4): 612-625 (April 1980).

16. McQuillan, John M. "Enhanced Message Addressing Capabilities for Computer Networks," Proceedings of the IEEE,66 (11): 1517-1527 (1978).

17. Sunshine, Carl A. and Yogen K. Dalal. "Connection Management in Transport Protocols."

18. Phister, Capt Paul W. Protocol Standards and Implementation Within the Digital Engineering Laboratory Computer (DELNET) Using the Universal Network Interface Device (UNID)<pI`and II. MS Thesis, GE/EE/83D-58. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB, OH, (December 1983).

19. Sauer, Charles H. "Queueing Network Simulations of Computer Communication," IEEE Journal on Selected Areas in Communicatons, SAC-2 (1): 203-220 (January 1984).

20. Sauer, Charles H. Simulation of ComputerCommunication Systems. Prentice Hall, Englewood Cliffs, NJ. (1983).

VITA

Captain Gail M. Nusz was born on 3 November 1956 in Chicago, Illinois. She graduated from high school in Villa Park, Illinois, in 1974 and attended the University of Illinois from which she received the degree of Bachelor of Science in Computer Science in May 1978. Upon graduation, she received a commission in the USAF through the OTS program. She served as a computer programmer assigned to the ASD Computer Center, Wright Patterson AFB, Ohio, from October 1978 through December 1979. She was then assigned to the 3900 Computer Services Squadron at Offutt AFB, Nebraska as a computer system analyst until entering the School of Engineering, Air Force Institute of Technology, in May 1983.

Permanent address: 125 S. Chase Ave
Lombard, Illinois 60148

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A174268

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release;
distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AFIT/GCS/ENG/84D-19

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
School of Engineering

6b. OFFICE SYMBOL *(If applicable)*
AFIT/ENG

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State and ZIP Code)*
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433

7b. ADDRESS *(City, State and ZIP Code)*

8a. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State and ZIP Code)*

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE *(Include Security Classification)*
See Box 19

12. PERSONAL AUTHOR(S)
Gail M. Nusz, B.S., Captain, USAF

13a. TYPE OF REPORT
MS Thesis

13b. TIME COVERED
FROM ________ TO ________

14. DATE OF REPORT *(Yr., Mo., Day)*
1984 December

15. PAGE COUNT
125

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Networks; Communication Networks; Mobile; Packet switching

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Title: ADDRESSING FOR MOBILE SUBSCRIBERS IN A PACKET-SWITCHING MULTI-NETWORK ENVIRONMENT

Thesis Chairman: Walter D. Seward, Major, USAF

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS □

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Walter D. Seward, Major, USAF

22b. TELEPHONE NUMBER *(Include Area Code)*
513-255-3576

22c. OFFICE SYMBOL
AFIT/ENG

DD FORM 1473, 83 APR

EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

This thesis addresses the problem of maintaining the TCP-based applications of a subscriber in a mobile environment. Two solutions to the problem of addressing these mobile subscribers has been suggested. The first scheme of addressing is to use logical addresses at the internet level and the second is to use separate addresses at the transport level of the protocol hierarchy. The performance of the two proposed solutions are analyzed through the use of a simulation program based on queueing models.

END

12-86

DTIC